Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет комп'ютерних систем і автоматики
(повне найменування інституту)
Кафедра метрології та промислової автоматики
(повна назва кафедри)

Пояснювальна записка

до магістерської кваліфікаційної роботи
магістр
(освітній ступень)

на тему Вдосконалення системи управління якості програмного забезпечення

Виконав: студент 2 курсу, групи ІЯП-18м
спеціальності 152 – Метрологія та інформаційно-вимірювальна техніка (освітня програма: інженерія якості продукції)
(шифр і назва спеціальності)

_____Петровський Д.Ю._____
(прізвище та ініціали)
Керівник Маньковська В.С._____
(прізвище та ініціали)
Рецензент Тарновський М.Г._____
(прізвище та ініціали)

Вінниця - 2019 року

Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет комп'ютерних систем і автоматики
Кафедра метрології та промислової автоматики
Освітній ступень ___магістр_____
Спеціальність 152 – Метрологія та інформаційно-вимірювальна техніка
(освітня програма: інженерія якості продукції)
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри МПА
_____ д.т.н., проф. Кучерук В.Ю.
"____" _____20___року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Петровському Дмитру Юрійовичу _____
(прізвище, ім'я, по батькові)

1. Тема роботи Вдосконалення системи управління якості програмного забезпечення
_____
_____
керівник роботи Маньковська Вікторія Сергіївна, к.т.н., доцент_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від "___"_____20__року №___

2. Строк подання студентом роботи_____

3. Вихідні дані до роботи етап життєвого циклу ПЗ – розробка та конструювання; застосувати вимірювальні та реєстраційні метрики; серед метрик використати цикломатичну складність; досліджуваний програмній продукт містить не більше 10000 рядків коду

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) провести огляд існуючих систем якості, проаналізувати методики та підходи до оцінки якості програмного забезпечення, визначити найбільш ефективні, визначити недоліки, запропонувати шляхи покращення, підтвердити справедливість зроблених припущень, сформулювати задачі подальшого дослідження

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Вдосконалення системи управління якості програмного забезпечення Технічне завдання на бакалаврську дипломну роботу; Вдосконалення системи управління якості програмного забезпечення Моделі життєвого циклу програмного забезпечення; Вдосконалення системи управління якості програмного забезпечення Показники якості та їх значущість

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Маньковська В.С., доцент кафедри МПА	02.09.2019	13.09.2019
2	Маньковська В.С., доцент кафедри МПА	13.09.2019	27.09.2019
3	Маньковська В.С., доцент кафедри МПА	27.09.2019	07.10.2019
4	Маньковська В.С., доцент кафедри МПА	07.10.2019	18.10.2019
5	Маньковська В.С., доцент кафедри МПА	18.10.2019	17.11.2019
6	Ратушняк О.Г., доцент кафедри ЕПВМ	18.11.2019	06.12.2019

7. Дата видачі завдання 02.09.2019 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Якість програмного забезпечення та методи її контролю	02.09.2019 13.09.2019	виконано
2	Визначення метрик якості програмного забезпечення на ранніх етапах життєвого циклу	13.09.2019 27.09.2019	виконано
3	Оцінка ефективності метрик якості на ранніх етапах життєвого циклу ПЗ	27.09.2019 07.10.2019	виконано
4	Оцінка результатів	07.10.2019 18.10.2019	виконано
5	Експериментальні дослідження	18.10.2019 17.11.2019	виконано
6	Економічна частина	18.11.2019 06.12.2019	виконано

Студент _____ Петровський Д.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Маньковська В.С.
(підпис) (прізвище та ініціали)

ЗМІСТ

РЕФЕРАТ

Магістерська кваліфікаційна робота присвячена пошуку шляхів вдосконалення системи управління якості програмного забезпечення. В роботі проводиться ґрунтовний огляд існуючих систем управління якості ПЗ, методів контролю якості та загальних підходів, що застосовуються для створення якісного програмного продукту. Визначається множина метрик для оцінки якості ПЗ на ранніх стадіях життєвого циклу.

REFERAT

Master's qualification is dedicated to finding ways to improve the software quality management system. The paper provides a thorough review of existing software quality management systems, quality control methods and general approaches used to create a quality software product. A set of metrics is defined to evaluate the quality of software in the early stages of the life cycle.

ВСТУП

Сьогодні галузь розробки програмного забезпечення (ПЗ) вважається найбільшою галуззю світової економіки. За деякими оцінками [1] на 2015 рік кількість розробників ПЗ у світі сягнула 19 мільйонів осіб, а до 2019 року прогнозується зростання їх кількості до 25 мільйонів. В Україні за попередніми оцінками [2] на 2016 рік було налічено 99940 осіб зайнятих в IT галузі, і до цієї кількості входять не тільки програмісти, а й інженери з якості ПЗ, менеджери проектів, тощо. Інша ж частина людства безпосередньо залежить від їх успішної діяльності.

За час становлення галузі, завдання, їх складність, форми подання даних та методи їх обробки сильно змінились, але до сьогодні розробка якісного програмного забезпечення не стала нормою [3, 4]. Очевидно, що в галузі забезпечення якості ПЗ – криза. Великі проекти виконуються із запізненням або перевищенням кошторису витрат, розроблені програми не відповідають функціональним вимогам, ПЗ має низьку продуктивність, а якість програм не влаштовує користувачів.

Останнім часом група The Standish Group кожні два роки публікує звіт «The Chaos Report» з аналізом результатів впровадження програмних проектів. Результати аналізу [5] показують, що у 2016 році лише 16,2 % проектів завершились з дотриманням графіку та вклалися в кошторис витрат. Беручи до уваги той факт, що США щорічно витрачає більше ніж 250 млрд. дол. на IT галузь в цілому [5], на успішні проекти витрачається лише 40,5 млрд. дол. Інші кошти витрачаються на проекти, які ніколи не будуть впроваджені і не принесуть користі.

Наразі розроблено чимало засобів, методів, стандартів та технологій, які покликані забезпечити якість програмного забезпечення, але якість ПЗ, як і раніше, залежить в основному від знань та досвіду розробників. Проекти не рідко зазнають невдач через невірне формулювання вимог, невдале проектування або неефективне планування, неправильне розуміння або

недостатній аналіз специфікації, і, як наслідок, помилки на ранніх етапах життєвого циклу програмного забезпечення. Тому, виявляється, що кращий спосіб підвищення якості ПЗ є мінімізація часу, який витрачається на виправлення коду, в наслідок зміни вимог, зміни проекту чи виконання відлагоджування. Це підтверджується реальними даними досліджень [6] проведених в лабораторіях NASA, які показали, що підвищена увага до контролю якості дозволяла знизити рівень помилок, але не підвищувала загальні витрати на розробку.

Проте процес розробки програмного забезпечення, як і процес оцінювання якості ПЗ залишаються такими, що вимагають системного підходу, що має наукове підґрунтя. Тому пошук шляхів покращення системи управління якістю програмного забезпечення залишається **актуальною** задачею.

Зв'язок роботи з науковими програмами, планами, темами. Теоретичні та експериментальні дослідження, результати яких отримані у магістерській кваліфікаційній роботі, виконувалися на кафедрі "Метрологія та промислова автоматика" Вінницького національного технічного університету.

Мета і задачі дослідження. Метою роботи є пошук шляхів вдосконалення системи управління якості програмного забезпечення в рамках існуючих методик та підходів до оцінки якості.

Для досягнення мети сформульовано наступні задачі:

- провести огляд існуючих систем якості;
- проаналізувати методики та підходи до оцінки якості програмного забезпечення;
- визначити найбільш ефективні, визначити недоліки, запропонувати шляхи покращення;
- підвищити ефективність методів та підходів до визначення якості ПЗ.

Методи дослідження. Для досягнення мети і вирішення поставлених

завдань використано теоретичні та експериментальні методи досліджень, що ґрунтуються на загальній теорії вимірювання, методах наукового пошуку, методах математичної статистики.

Об'єктом роботи є система управління якості програмного забезпечення.

Наукова новизна отриманих результатів. В роботі отримані наступні наукові результати:

1. Вдосконалено існуючі системи управління якості програмного забезпечення;

2. Підвищено ефективність методів та підходів до визначення якості ПЗ.

Практичне значення одержаних в роботі результатів полягає в тому, що:

1. в результаті аналізу існуючих методів і підходів визначені найбільш слабкі місця в сучасній системі управління якістю програмного забезпечення;
2. запропонована множина метрик якості та діапазони їх зміни для контролю якості ПЗ на ранніх етапах життєвого циклу.

Особистий внесок здобувача. Основні наукові результати, які були отримані під час написання магістерської кваліфікаційної роботи отримані автором одноосібно.

1 ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА МЕТОДИ ЇЇ КОНТРОЛЮ

Шукаючи відповідь на питання «що таке якість програмного забезпечення?» важливо розуміти, що наразі не існує одного ємкого визначення, яке б характеризувало цю категорію зі всіх боків. Проте, для більшості є очевидним, що якісне програмне забезпечення, це продукт:

- який легко використовувати;
- який демонструє високу продуктивність;
- в якому немає помилок;
- який не псує даних користувача при збоях;
- який можна використовувати на різних платформах;
- який може працювати 24 години на добу 7 днів в тижні;
- до якого легко додати нові можливості;
- який задовольняє вимогам користувача;
- який добре задокументований.

Однак спроби сформулювати поняття якості ПЗ все ж відбувались протягом становлення ІТ галузі.

Так, якість за [7] визначається як здатність програмного продукту при заданих умовах задовольняти встановленим вимогам, або вимогам, що передбачаються.

Інші нормативні документи визначають якість ПЗ як наприклад:

- весь об’єм ознак та характеристик програм, який відносять до їх здатності задовольняти встановленим потребам або, тим які передбачаються [8];

- ступень, в якому компонента, система або процес задовольняють потребам або очікуванням замовника або користувача [9].

Але всі спроби окреслити загальну концепцію зводяться до того, що якість визначається як сукупність властивостей, що забезпечують задовільний рівень сприйняття програмного продукту користувачем, а для

формування кількісної оцінки повинні визначатись критерії оцінки якості ПЗ.

На разі критеріями якості ПЗ можна вважати:

- функціональність (здатність ПЗ виконувати набір функцій, які задовольняють потребам користувачів. Набір зазначених функцій визначається в зовнішньому описі ПЗ);
- надійність (це здатність ПЗ безвідмовно виконувати визначені функції при заданих умовах протягом заданого періоду часу з досить великою імовірністю);
- легкість застосування (характеристика ПЗ, яка дозволяє мінімізувати зусилля користувача по підготовці вхідних даних, застосуванню ПЗ і оцінці отриманих результатів);
- ефективність (відношення рівня послуг, які надає ПЗ користувачу при заданих умовах, до обсягу використовуваних ресурсів);
- супровід (характеристики ПЗ, що дозволяють мінімізувати зусилля по внесенню змін для усунення в ньому помилок і по його модифікації відповідно до потреби користувача);
- мобільність (здатність ПЗ бути перенесеним з одного середовища (оточення) в інше, зокрема, з одного комп'ютера на інший).

Причому функціональність та надійність є обов'язковими критеріями якості ПЗ, а надійність до того ж критерій, який застосовується на всіх етапах розробки ПЗ.

З вище наведеного можна зробити висновок про те, що якість загалом, і зокрема якість програмного забезпечення поняття динамічне і може змінюватись оскільки з часом змінюються і потреби споживачів, тому постійно необхідно корегувати і вимоги до якості. Також необхідно брати до у ваги той факт, що якість – це сукупність властивостей, для опису яких необхідно мати певним чином визначені показники, а для оцінки цих властивостей – кількісні характеристик. Відправною точкою в процесі формування цієї сукупності динамічно змінюваних властивостей можна вважати сукупність нормативних документів серії ISO 9126 [10-14].

1.1 Основні міжнародні стандарти в області якості програмних засобів

Однією з найважливіших проблем забезпечення якості програмних засобів є формалізація характеристик якості і методологія їх оцінки. Для визначення адекватності якості функціонування, наявність технічних можливостей програмних засобів до взаємодії, вдосконалення і розвитку необхідно використовувати стандарти в області оцінки характеристик їх якості [15]. Модель формування програмного продукту з огляду на дотримання основних вимог нормативних документів зображена на рисунку 1.1.

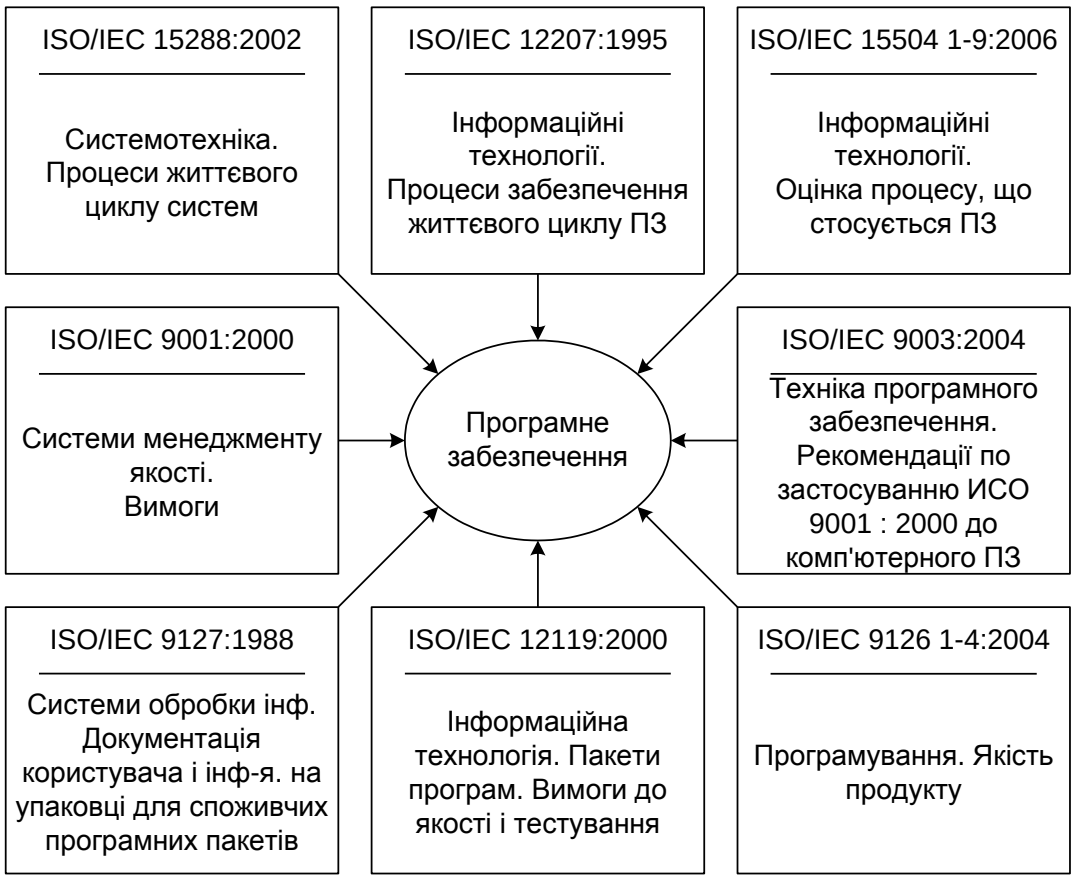

Рисунок 1.1 – Основні міжнародні стандарти в області програмних засобів

Перша частина стандарту ISO 9126-1 описує характеристики якості програмного забезпечення, що використовуються в інших частинах стандарту. Виходячи з принципових можливостей їх вимірювання, усі характеристики якості можуть бути об'єднані в три групи, до яких застосовані різні категорії метрик [15]:

- метрики категорій, або описові (номінальні) метрики найбільш адекватні для оцінки функціональних можливостей програмних засобів;
- кількісні метрики, що придатні для вимірювання надійності і ефективності складних комплексів програм;
- якісні метрики найбільшою мірою відповідають практичності, можливості простого супроводу і мобільності програмних засобів.

Друга і третя частини стандарту присвячені формалізації відповідно зовнішніх і внутрішніх метрик характеристик якості складних програмних засобів. Четверта частина стандарту ISO 9126-4 призначена для покупців, постачальників, розробників, супроводжуючих користувачів і менеджерів якості програмних засобів. В цій частині стандарту обґрунтовуються і коментуються виділені показники сфери (контексту) використання програмних засобів і групи вибраних метрик для користувачів.

Вихідними даними і вищим пріоритетом при виборі показників якості у більшості випадків є призначення, функції і функціональна придатність відповідного програмного засобу. Досить повний і коректний опис цих властивостей повинен бути базою для визначення значень більшості інших характеристик і атрибутів якості. Принципові і технічні можливості і точність вимірювання значень атрибутів характеристик якості завжди обмежені відповідно до їх змісту. Це визначає раціональні діапазони значень кожного атрибуту, які можуть бути вибрані на основі здорового глузду, а також шляхом аналізу прецедентів в специфікаціях вимог реальних проектів.

Як наслідок ретельного аналізу та формалізації характеристик якості програмного забезпечення в нормативних документах, що згадані віще,

виділяють шість характеристик якості, які формують узагальнену модель якості. На верхньому рівні (рисунок 1.2) виділені основні характеристики якості ПЗ. Кожна характеристика описується за допомогою декількох атрибутів, що входять в неї.

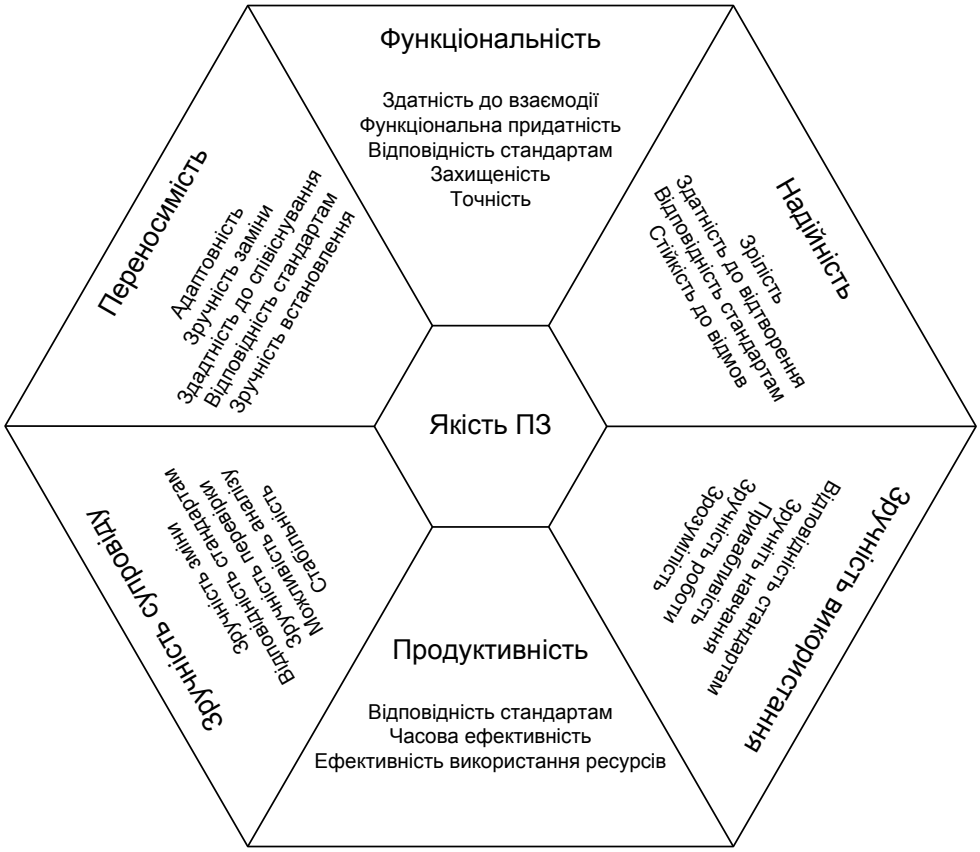

Рисунок 1.2 – Модель якості програмного забезпечення
за ISO 9126 1-4:2001-2004

Кожна з наведених характеристик описується сукупністю показників і атрибутів. Перелік цих показників і атрибутів до кожної характеристики зведений до таблиці 1.1.

Таблиця 1.1 – Опис характеристик якості

№	Характеристика якості	Опис характеристики якості
1	2	3
1	Функціональність	Здатність ПЗ в певних умовах вирішувати завдання, потрібні користувачам. -Функціональна придатність (suitability). -Точність (accuracy). -Здатність до взаємодії (interoperability). -Відповідність стандартам і правилам (compliance). -Захищеність (security).
2	Надійність	Здатність ПЗ підтримувати певну працездатність в заданих умовах. -Зрілість, завершеність (maturity). -Стійкість до відмов (fault tolerance). -Здатність до відновлення (recoverability). -Відповідність стандартам надійності (reliability compliance).
3	Зручність використання	Здатність ПЗ бути зручним в навчанні і використанні, а також привабливим для користувачів. -Зрозумілість (understandability). -Зручність навчання (learnability). -Зручність роботи (operability). -Привабливість (attractiveness). -Відповідність стандартам зручності використання (usability compliance).
4	Продуктивність	Здатність ПЗ за заданих умов забезпечувати необхідну працездатність по відношенню до ресурсів, що виділяються для цього. -Тимчасова ефективність (time behaviour). -Ефективність використання ресурсів (resource utilisation). -Відповідність стандартам продуктивності (efficiency compliance).
5	Зручність супроводу	Зручність проведення усіх видів діяльності, пов’язаних з супровід програм. -Здатність до аналізу (analyzability) або зручність проведення аналізу. -Зручність внесення змін (changeability). -Стабільність (stability). -Зручність перевірки (testability). -Відповідність стандартам зручності супроводу (maintainability compliance).

Продовження таблиці 1.1

1	2	3
6	Переносимість	Здатність ПЗ зберігати працездатність при перенесенні з одного оточення в інше, включаючи організаційні, апаратні і програмні аспекти оточення. -Здатність до адаптації (adaptability). -Продуктивність (productivity). -Безпека (safety). -Задоволення користувачів (satisfaction).

Приведені атрибути якості закріплені в стандартах, але це не означає, що вони повністю вичерпують поняття якості ПЗ [16]. Так, в стандарті ISO 9126 повністю відсутні характеристики, пов’язані з мобільністю ПЗ (mobility), тобто здатністю програми працювати при фізичних переміщеннях машини, на якій вона працює. Замість надійності багато дослідників вважають за краще розглядати більш загальне поняття добротності (dependability), що описує здатність ПО підтримувати певні показники якості по основних характеристиках (функціональності, продуктивності, зручності використання) із заданою вірогідністю виходу за їх рамки і заданими ризиками можливих порушень. Крім того, активно досліджуються поняття зручності використання, безпеці і захищеності ПО – вони здаються більшості фахівців набагато складнішими, ніж це описується даним стандартом.

Для оцінки атрибутів притаманних кожній характеристиці якості існує поняття метрики – це міра, що дозволяє отримати числове значення деяких властивостей програмного забезпечення та його специфікацій [17]. Кількісні методи оцінювання добре показали себе в інших сферах науки, а тому багато теоретиків та практиків в галузі інформаційних технологій, спробували перенести цей підхід в розробку програмного забезпечення. В загальному випадку застосування метрик дозволяє визначити складність розробленого проекту, або проекту, що перебуває у розробці, оцінити об'єм робіт, стилістику розроблюваного проекту і зусилля, витрачені кожним розробником для реалізації того чи іншого рішення, однак метрики можуть

служити лише рекомендаційними характеристиками.

До основних метрик оцінки атрибутів згідно ISO 9126 1-4:2001-2004 відносять:

- повнота реалізації функцій;
- коректність реалізації функцій;
- відношення числа знайдених дефектів до числа прогнозованих;
- відношення числа проведених тестів до повної їх кількості;
- відношення числа доступних проектних документів до вказаної їх кількості в списку;
- наочність та повнота документації.

1.2 Методи контролю якості ПЗ

До методів контролю якості програмного забезпечення, серед іншого, згідно [18] відносять:

- інспекції (огляди);
- тестування програмного забезпечення;
- статичний аналіз програмного забезпечення;
- імовірнісна оцінка надійності програмного забезпечення;
- аналіз видів, наслідків і критичності відмов програмного забезпечення.

1.2.1 Інспекції програмного забезпечення

Інспекція програмного забезпечення [19] це аналіз та перевірка різних робочих продуктів ПЗ (специфікацій, архітектурних схем, діаграм, вихідного коду та ін.) і виконується на всіх етапах життєвого циклу розробки програмного забезпечення. Метою інспекції є виявлення різних аномальних станів програмного забезпечення незалежними фахівцями та з залученням авторів проміжного або кінцевого продукту.

Особливістю застосування інспекції є те, що для її виконання немає потреби в робочій програмі. Тому інспекція програмного забезпечення дозволяє виявляти дефекти на ранніх стадіях життєвого циклу. До того ж інспекція дозволяє виявляти дефекти в таких робочих продуктах як: плани тестування, керівництвах користувача, графіки проектів, тощо.

Узагальнена схеми процесу інспекції представлена на рисунку 1.3. Спираючись на приведену схему можна оцінити необхідні заходи для виявлення та усунення дефектів на ранніх стадіях життєвого циклу програмного забезпечення. В першу чергу для досягнення мети необхідно грамотно запланувати інспекцію (огляд), сформувати мету, визначитись з об’єктом інспекції, визначитись з тим, хто буде проводити інспекцію, хто буде залучений в процесі проведення оглядів.

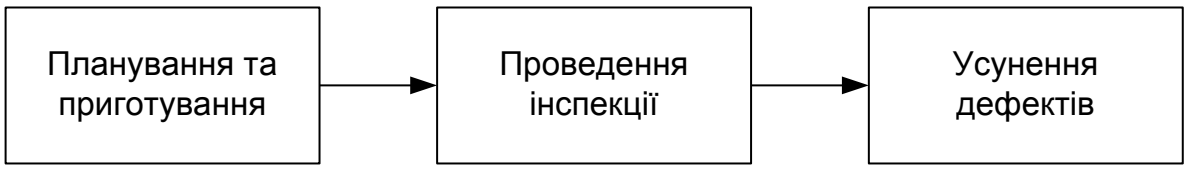

Рисунок 1.3 – Узагальнена схема процесу інспекції

Існує безліч способів перевірки якості, один з яких в 1970-х роках був запропонований Майклом Фаганом. Його метод статичного контролю якості в першу чергу торкався перевірки коду програмного забезпечення.

Мета інспекції Фагана – переконатися, що уся документація складена правильно і зрозуміло, відповідає прийнятим стандартам (зазвичай перевіряють тест кейси, коди, специфікації, плани тестування і таке інше). Такий метод перевірки якості передбачає наявність досить жорстких організаційних заходів, деякі навіть можуть стверджувати, що ці заходи носять бюрократичний характер.

Проте, в результаті впровадження інспекції Фагана, компанія по тестуванню може досягти більш високого рівня продуктивності, зменшення

кількості помилок вже на підготовчому етапі будь-якого виду тестування (автоматизоване тестування, тестування навантаження, тестування безпеки).

У інспекції може брати участь досить велика кількість людей. Вони повинні досягти консенсусу при затвердженні питань, пов'язаних з програмним продуктом, затвердити усю необхідну документацію і погоджувати усі особливості майбутнього проекту. Мета інспекції - виявити помилки в наявній документації.

Робочі ролі при проведенні інспекції Майкла Фагана [20] розподіляються наступним чином:

автор – людина, яка створює продукт, який в майбутньому інспектуватиметься;

модератор – головна особа в інспекції, що проводиться. Складає план перевірки, є координатором усієї робочої групи. Також він виконує функції голови засідань, на яких наважуються питання про реєстрацію дефектів;

читач – роль цієї людини зводиться лише до того, що він зачитує перед іншими членами комісії текст документів, які будуть проінспектовані;

реєстратор – особа, яка документує усі помилки, які були знайдені робочою групою. Роль реєстратора може виконувати тестувальник програмного забезпечення. Реєстратор також складає список питань, які не розглядаються під час сеансу тестування. Ці питання можуть включати не лише знайдені несправності. Іноді реєстратор може передавати інформацію від робочої групи іншим учасникам проекту, а також членам групи, які з яких-небудь причин були відсутні під час інспекції;

інспектор – повинен дати об'єктивну оцінку програмного продукту, як з призначеної для користувача, так і з технічної точки зору. Ця людина перевіряє робочий продукт з метою виявлення можливих дефектів. Інспектор несе особисту відповідальність за якість продукту, що перевіряється. У робочій групі може бути декілька інспекторів.

Інспекція за Фаганом складається з шести етапів і може бути представлена схемою зображеною на рисунку 1.4.

До інспекції залучається як правило чотири інспектора, яких визначають на стадії планування серед досвідчених інженерів, які знайомі з об’єктом інспекції проте самі не задіяні в його розробці.

Аналізуючи процедуру інспекції можна сформулювати наступні тезі, які є основою якості процесу інспектування:

- важливість підготовки процесу інспектування;
- важливість гнучкості розміру команди інспекторів;
- систематичні прийоми виявлення дефектів ефективніші за ситуативні;
- важливим є застосування зворотного зв’язку інспектування.

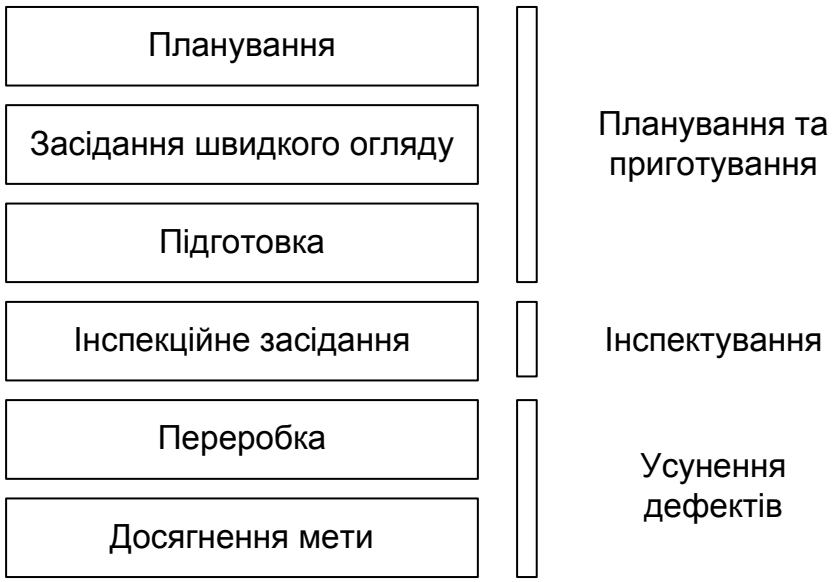

Рисунок 1.4 – Етапи проведення інспекції за Фаганом

Складно переоцінити процедуру інспектування, проте вона вимагає дуже великих затрат часу та зусиль на проведення. Тому в реальних умовах виробництва часто застосовують спрощену процедуру (two-person inspection). Така процедура є спрощеною процедурою за Фаганом, але містить в собі всі її основні етапи і широко застосовується при ітеративному підході розробки програмного забезпечення.

Іще одним варіантом інспекції є інспекція без засідань. Оскільки на

засіданнях виявляють лише 5-30 % дефектів і засідання вимагають витрати великої кількості ресурсів. Інспекція без засідань зменшує витрати не набагато зменшуючи ефективність. Проте недоліком такої інспекції є те, що така інспекція збільшує кількість помилкових тривог.

Ще одним варіантом інспекції є інспекція за Глібом особливостями якої є:

- входом процесу інспекції є документи правила, контрольні переліки. Будь-який технічний документ може бути проінспектований;
- виходом є виправлені вхідні документи, та пропозиції по вдосконаленню процесу;
- процес інспектування формує цикл із зворотнім зв’язком.

На інспекційному засіданні інспектори прагнуть окрім виявлення дефектів з'ясувати їх причини та запропонувати виправлення. Гліб вирішує проблему шляхом додавання до процесу інспекції крок «процес мозкового штурму». У центрі уваги цього кроку є аналіз причин, спрямований на вироблення профілактичних заходів для зниження ін’єкцій дефектів у ПЗ.

Схематично інспекцію за Глібом можна представити у вигляді схеми зображеної на рисунку 1.5.

Також як варіант інспекції використовують так звану «перевірку за столом» (desck check) - неформальна перевірка технічних документів, що створена перевіряльником (самоперевірка), для виправлення очевидних помилок. Фокус робиться на логічних та концептуальних помилках, які виправляються на стадії виявлення. Доповнюють «перевірки за столом» процесом рецензування (review) - неформальною перевіркою технічних документів, яка створена кимось іншим. Фокус робиться на логічних та концептуальних помилках. Такі перевірки виконують як правило індивідуально або групами.

Більш організована форма перевірок для програмного коду та моделей називається «проходженням» (walkthrough).

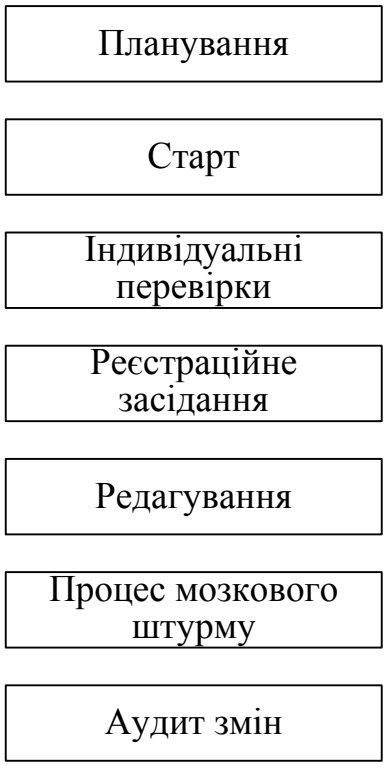

Рисунок 1.5 – Етапи проведення інспекції за Глібом

Такий підхід до організації інспекції виконують на засіданнях де головує автор. Основою процесу проходження складає імітування програми (перевірка алгоритмів на предмет вирішення поставленої задачі).

Часто при інспектуванні коду застосовують формальні підходи – читання з покроковим абстрагуванням. Така практика спрямована на реалізацію процедури декомпозиції і дозволяє сфокусуватись на частинах програми та абстрагуватись від частин коду більш низького рівню.

Перелічені підходи до організації інспекції дозволяють в першу чергу виявляти дефекти. Цікаво, що на практиці дефекти можна виявити спонтанно. Хоча для виявлення основної маси дефектів використовують так звані контрольні списки або виявляють дефекти на основі сценаріїв.

При використанні різних контрольних списків для гарантії покриття важливих частин документів як правило використовують: контрольні списки по робочим продуктам (перевіряються основні функції, структури даних, оголошення компонентів); контрольні списки за властивостями

(перевіряються стиль коду, відповідність стандартам, зв'язаність та залежність модулів коду).

1.2.2 Тестування програмного забезпечення

Тестування програмного забезпечення [21] (*англ.* Software Testing) – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись. Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою оцінки. Може оцінюватись:

- відповідність вимогам, якими керувалися проектувальники та розробники;
- правильна відповідь для усіх можливих вхідних даних;
- виконання функцій за прийнятний час;
- практичність;
- сумісність з програмним забезпеченням та операційними системами;
- відповідність задачам замовника.

Поняття тестування ПЗ тісно пов’язане з поняттям життєвого циклу ПЗ. Життєвий цикл програмного забезпечення (SDLC – Software Development Life Cicle) – період часу, який починається з моменту затвердження рішення про створення програмного продукту і завершується тоді, коли програмний продукт виводять з експлуатації. Типовий цикл складається з восьми основних етапів: планування; аналіз вимог; дизайн і розробка; впровадження; тестування; оцінка; реліз; підтримка. У більш широкому сенсі тестування – це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Основною метою процесу тестування – є доказ того, що результат розробки відповідає пред’явленим до нього вимогам. Основне завдання

тестування ПЗ: отримання інформації про статус готовності заявленої функціональності системи або програми.

Тести істотно різняться за завданням, які з їх допомогою вирішуються, і за технікою, що використовується. Відмінність завдань тестування призводить до необхідності використати різноманітні типи (види) тестування. Прийнято підрозділяти тестування на види по наступних категоріях: об’єкти (елементи) тестування, часто розділення на види тестів за цим критерієм називають розділенням тестування на рівні; за глибиною тестування, тобто розділення тестових випробувань на типи проводиться залежно від кількості часу і об’єму компонент програмного продукту, що тестується. Проте, основна класифікація тестів на види робиться у відповідність з традиційними показниками якості, які перевіряються з їх допомогою.

Модульне тестування (Unit Testing). Вхідними вимогами для такого тестування є архітектура компонентів або модель «нижнього рівня» системи (Component Design, Low Level Design). Об'єктом тестування є розроблені компоненти. На цьому рівні тестуються окремо невеликі елементи системи, максимально відокремлені від інших елементів, які одночасно придатні для тестування. Таке тестування зазвичай проводиться відразу після розробки кожного з елементів і спрямовано на оцінку відповідності функціональності кожного з компонентів спроектованої «моделі компонентів».

Комплексне тестування (Integration Testing, Interface Testing). Вхідними вимогами для такого виду тестування є архітектура системи або модель «верхнього рівня» системи (System Design, High Level Design). Об'єктом тестування є зібрана з компонентів система або підсистема. На цьому рівні тестуються об’єднані елементи (компоненти або підсистеми) загальної системи, найчастіше деяка група елементів, що взаємодіє між собою. Комплексне тестування спрямоване не на перевірку функціонування кожного з компонентів, а на перевірку взаємодії компонентів відповідно до «архітектури системи». Тести на цьому рівні зазвичай перевіряють усі

інтерфейси взаємодії між компонентами, визначені в системній архітектурі, до тих пір, поки усі компоненти не будуть розроблені, відлагоджені і об’єднані в єдину систему.

Системне тестування (System Testing). Вхідними вимогами для цього виду тестування є системні специфікації (System Specification). Об'єктом тестування є розроблена система. Після того, як система зібрана з складових компонентів, вона має бути протестована на відповідність «системним специфікаціям» та на реалізованість усіх функціональних і нефункціональних вимог до системи, що розробляється.

Приймальне тестування (Acceptance Testing). Вхідними вимогами є вимоги до програмного продукту (Requirements). Об'єктом тестування виступає розроблена система. На цьому рівні завершена програма тестується замовником, кінцевими користувачами або відповідними уповноваженими з метою визначення відповідності системи «вимогам замовника» і готовності системи до впровадження. Приймально-здавальні випробування оформляють процес передачі продукту від розробника до замовникові. Залежно від особливостей продукту і від вимог замовника вони можуть проводитися в різній формі. Наприклад, у виді альфа- або бета-тестування. Приймальне тестування схоже з системним тестуванням, але з наступною відмінністю: приймальне тестування перевіряє, чи розроблена система задовольняє вимогам замовника з наголосом на потреби кінцевих користувачів в цільовій предметній області.

Операційне тестування (Release Testing). Вхідними вимогами є бізнес модель (Business Case, Business Model). Об'єктом тестування є розроблена система. Навіть, якщо система задовольняє усім вимогам, важливо переконатися в тому, що вона задовольняє потребам користувача і виконує свою роль в середовищі своєї експлуатації, як це було визначено у бізнес-моделі системи. Слід врахувати, що і бізнес модель може містити помилки. Тому так важливо провести операційне тестування як фінальний крок валідації.

Процес тестування також можна класифікувати за видами.

Інсталяційне тестування (Installation Testing). В процесі інсталяційного тестування перевіряється коректність установки і деінсталяції програмного продукту в середовищі максимально наближеному до того, в якому програмний продукт буде працювати. Перевірка правильності установки програмного продукту має бути обов'язковим елементом проекту по тестуванню будь-якого продукту. Мета такого виду тестування полягає в тому, щоб переконатися, що продукт може бути встановлений/деінстальований за різних умов – таких як: нова інсталяція, удосконалення системи (upgrade), установка за замовчанням, повна установка, установка за вибором.

Димне тестування (Smoke Testing). Перший прогін програми (після написання або після внесення істотних змін). Як правило, використовується для визначення, чи готова програма для проведення більшого тестування. Метою є виявлення проблем що «лежать на поверхні», тому тестується найчастіше основна бізнес логіка програми.

Функціональне тестування (Functional testing). Перевірка відповідності продукту функціональним вимогам і специфікаціям. Метою є перевірка відповідності продукту функціональним вимогам і специфікаціям.

Регресійне тестування (Regression Testing). Повторне тестування після внесення змін до програмного забезпечення або в його оточення, щоб переконатися в тому, що функції, які працювали в попередній версії системи, як і раніше працюють так, як очікувалося, а знайдені дефекти успішно виправлені. Метою є виявлення потенційних проблем, які могли виникнути в результаті змін.

Інтеграційне тестування (Integration Testing). Перевірка скомбінованих компонентів прикладної програми з метою визначення коректності їх спільного функціонування. Метою є виявлення потенційних проблем в спільному функціонуванні компонентів.

Тестування графічного інтерфейсу користувача (User Interface Testing)

Тестування інтерфейсу – вікон, кнопок і таке інше. Велика частина функціональності ПЗ реалізується, як правило, через призначений для користувача інтерфейс. Метою є виявлення помилок в інтерфейсі і пошук помилок у функціональності за допомогою інтерфейсу.

Тестування продуктивності (Performance Testing). Перевірка швидкодії роботи системи (час відгуку, частота транзакцій і інші залежні від часу параметри) в імітаційній і реальній середовищах. Метою є встановлення реальної продуктивності програмного продукту.

Тестування навантаженням (Load Testing). Це ті ж тести продуктивності, при яких система піддається різним навантаженням; при цьому мета цього тестування – оцінити здатність системи правильно функціонувати при деякому перевищенні планованих навантажень при реальній експлуатації (система має деякий «запас міцності»). За мету ставиться задача переконатися в тому, що система працює відповідно до очікуваних робочих параметрів навантаження (оцінити межу працездатності).

Стрес тестування (Stress Testing). Є одним з різновидів тестування на продуктивність. Перевіряється поведінка системи за умови браку ресурсів (дискового простору, обривів мережі і таке інше). Метою є перевірка того, що система адекватно реагує на ті або інші стресові ситуації.

Конфігураційне тестування (Configuration Testing). Конфігураційне тестування – перевірка роботи на різних платформах. Різні варіанти апаратної конфігурації, версії операційної системи і оточення. Метою є перевірити працездатність системи при різних конфігураціях.

Тестування інтернаціоналізації (Internationalization Testing). Цей вид тесту визначає наскільки продукт готовий до того, щоб бути адаптованим для роботи в інших локалізаціях: іншою мовою призначеного для користувача інтерфейсу, відмінному від мови за умовчанням (як правило, це англійський). Метою є перевірка здатності продукту бути швидко зміненим під необхідну локалізацію потенційних користувачів системи.

Тестування надійності (Reliability Testing). Цей вид перевірки проводиться з метою визначення нефункціональних вимог, чи програма працює, як і очікувалося, є стійкою до порушень в роботі і тому подібне. В даному випадку застосовуються інтеграційні тести, тести структури, стресові тести і інші.

Тестування зручності використання (Usability Testing) Проводиться з метою упевнитися в тому, що програма зручна для використання її кінцевим користувачам. Включає тести на людський чинник, естетику інтерфейсу і його несуперечність, наявність і якість оперативної і контекстної допомоги, керівництва користувача і навчальних матеріалів.

Таблиця 1.2 – Показники ефективності різних методів тестування

Назва методу	Мінімальна ефективність	Середня ефективність	Максимальна ефективність
Персональний перегляд проектних документів	15%	35%	70%
Неформальні групові перегляди	30%	40%	60%
Формальні перегляди проектних документів	35%	55%	75%
Формальні інспекції коду	30%	60%	70%
Моделювання та протитипування	35%	65%	80%
Перевірка за партою	20%	40%	60%
Тестування модулів	10%	25%	50%
Функціональне тестування	20%	35%	55%
Комплексне тестування	25%	45%	60%
Тестування в реальних умовах	35%	50%	65%
Застосування всіх перелічених методик	**93%**	**99%**	**99%**

Тестування продуктивності (Performance Testing). Виконується з метою упевнитися, що функціонування програми забезпечується в той час, коли виконуються нефункціональні вимоги до програми при роботі в реальних умовах. Включає оцінку тимчасових профілів, часу відгуку, операційної надійності і деяких інших характеристик.

Очевидно, що підходів до тестування і методів, які застосовуються в процесі перевірки якості програмного забезпечення велика кількість і всі вони мають кінцеву ефективність. В таблиці 1.2 приведені показники ефективності методів тестування (мінімальний, середні та максимальний).

1.2.3 Статичний аналіз програмного забезпечення

Статичний аналіз коду [22] (*англ.* Static Code Analysis) – аналіз програмного забезпечення, що виконується (на відміну від динамічного аналізу) без реального виконання досліджуваних програм. У більшості випадків аналіз проводиться над будь-якою версією початкового коду, хоча іноді аналізу підлягає який-небудь вид об'єктного коду, наприклад Р-код або код на MSIL. Термін зазвичай застосовують до аналізу, вироблюваного спеціальним програмним забезпеченням (ПЗ), тоді як ручний аналіз називають «program understanding», «program comprehension» (розумінням або досягненням програми).

З іншого боку [23] статичним аналізом можна вважати групу методів, що використовує вихідний код програми для вилучення необхідної інформації.

В системі контролю якості від статичного аналізу очікують виконання наступного переліку задач:

- оптимізація програми;
- перетворення програми;
- обфускація/деобфускація;
- визначення помилок.

Кожна з перелічених задач так чи інакше впливає на якість програмного забезпечення в цілому, і вирішується на ранніх стадіях життєвого циклу ПЗ, здебільшого на стадії розробки.

Але треба розуміти, що при виконанні задачі статичного аналізу коду можливо не тільки покращити якість в цілому, але й досягти погіршення результату. Наприклад, в процесі обфускації [24] з метою захисту від несанкціонованої модифікації, логічна структура програми стає неочевидною, що в свою чергу ускладнює роботу спеціалістів на стадії тестування із застосуванням методів «білого ящика».

Для досягнення мети при виконанні задач статичного аналізу програмного забезпечення виділяють наступні види статичного аналізу:

- пошук за шаблонами;
- аналіз потоку даних;
- аналіз потоку управління;
- аналіз паралельного виконання програми;

Як правило при виконанні статичного аналізу коду використовують всі види аналізу, що підвищує ефективність в цілому. Ефективністю статичного аналізу прийнято вважати властивість, якість результатів, що отримують, ступінь автоматизації аналізу та складність його організації, ресурсоємність та можливість застосування для будь-якої програми.

1.2.4 Імовірнісна оцінка надійності програмного забезпечення

Надійність програмного забезпечення набагато важливіше інших його характеристик, наприклад, часу виконання, і хоча абсолютна надійність сучасного програмного забезпечення, мабуть, недосяжна, досі не існує загальноприйнятої міри надійності комп'ютерних програм.

Проблема надійності програмного забезпечення відноситься, схоже, до категорії "вічних". У присвяченій їй монографії Г. Майерса [25], яка випущена в 1980 році, відзначається, що, хоча це питання розглядалося ще на

початку застосування обчислювальних машин, в 1952 році, він не втратив актуальності до теперішнього часу. Відношення до проблеми досить виразно сформульоване в [26]: «Надійність програмного забезпечення – безпритульне дитя обчислювальної техніки».

Надійність – поняття статистичне, тобто передбачається наявність деякої (досить великого) кількості однакових зразків, випробувань, тощо. Очевидно також, що є елемент випадковості. Вивченню випадкових явищ присвячений спеціальний розділ математики: теорія імовірності. Основне поняття цієї теорії – простір елементарних подій (вибірковий простір, простір результатів), на якому задається деяка (імовірнісна) міра. Випадкова величина, згідно теорії, є функція, задана на просторі елементарних подій. Нарешті, як міра надійності використовуються деякі характеристики випадкової величини (як правило, математичне очікування).

Перш ніж говорити про надійність об’єкту, слід уточнити, що мається на увазі під об’єктом. Як відомо, комп’ютерна програма має декілька різних форм: зовнішні специфікації, початковий текст, виконуваний код і так далі. Загальноприйнята точка зору полягає в тому, що програма є об’єктом, інваріантним відносно форм його представлення. Згідно з цією точкою зору, зовнішні специфікації, початкові тексти на мовах різних рівнів, а також виконувані коди для різних процесорів є різні форми представлення однієї і тієї ж програми. Вказана точка зору корисна при розробці програмного забезпечення, оскільки дозволяє виявити найбільш суттєві для додатка властивості програми, загальні для усіх її представлень, проте вона малопродуктивна, якщо йдеться, наприклад, про таку кількісну характеристику, як час виконання: ясно, що вказана характеристика відноситься лише до однієї з форм представлення – коду, що виконується і, крім того, залежить не лише від програма, але і від тип процесор.

Програма вважається правильною, якщо вона не містить помилок. Така програма не дає невірних результатів, тобто вона абсолютно надійна. Цей факт породив неправдиве уявлення про те, що число помилок в програмі

можна вважати найбільш природною мірою надійності [25]. Було виконано досить багато робіт, в яких пропонувалися різні методи оцінки числа помилок, що залишилися в програмі, за результатами її тестування, у тому числі метод «засмічення» відомими помилками, проте, як показують міркування, що наводяться нижче, кількість помилок в програмі не має ніякого відношення до її надійності:

1. Число помилок в програмі – величина «не спостережувана», спостерігаються не самі помилки, а результат їх прояву.

2. невірне спрацьовування програми може бути слідством не одній, а відразу декількох помилок.

3. Помилки можуть компенсувати один одного, так що після виправлення якоїсь однієї помилки програма може почати «працювати гірше».

4. Надійність характеризує частоту прояву помилок, але не їх кількість; в той же час добре відомо, що помилки проявляються з різною частотою: деякі помилки залишаються невиявленими після багатьох місяців і навіть років експлуатації, але, з іншого боку, неважко навести приклади, коли одна єдина помилка призводить до невірного спрацьовування програми при будь-яких початкових даних, тобто до нульової надійності.

Розглянемо класичну імовірнісну модель послідовності випробувань Бернулли. Простір елементарних подій в цій моделі містить 2^n точок, де n – число випробувань (в даному випадку під випробуванням мається на увазі запуск програми). Кожен запуск програми має два результати: правильний і неправильний. Позначимо вірогідність неправильного результату p, а вірогідність правильного – $(1-p)$. Вірогідність того, що з n запусків K приведуть до неправильного результату, виражається добре відомою формулою біномного розподілу [27].

$$B(p,n,k) = C(n,k) * pk * (1-p)(n-k), \qquad (1.1)$$

де C(n, k) – число поєднань. Вірогідність p апріорі невідома, але за результатами запусків відомі n і k. Величина B як функція p має максимум при

$$p = k/n. \quad (1.2)$$

В якості міри надійності програми можна прийняти величину

$$R = 1 - k/n = (n-k)/n, \quad (1.3)$$

значення якої (від 0 до 1) узгоджуються із загальноприйнятим сенсом терміну надійність: наприклад, якщо усі запуски закінчилися з помилковим результатом ($k = n$), то надійність – нульова.

Найбільш суттєве припущення в цій моделі полягає в тому, що запуски програми вважаються незалежними. Це означає, що результати попередніх запусків не дають ніякої інформації про результати наступного. Ясно, що це припущення на практиці виконується не завжди: наприклад, повторний запуск з тими ж вхідними даними дасть, очевидно, той же самий результат.

З формули (3) виходить, що оцінка надійності програми росте зі збільшенням числа її запусків за гіперболічним законом. Це підтверджує інтуїтивно зрозуміле міркування про те, що програма тим надійніше, чим більше досвіду її експлуатації, який залежить як від інтенсивності використання програми, так і від накладу комп'ютера, на якому вона запускається. Таким чином, надійність програм для персональних комп'ютерів типу IBM PC, загальний наклад яких складає нині близько 100 мільйонів, на декілька порядків вище за аналогічні програми для спеціалізованих процесорів (якщо, звичайно, такі програми дійсно існують і експлуатуються).

1.2.5 Аналіз видів, наслідків і критичності відмов ПЗ

Головними причинами, що викликають порушення нормального

функціонування ПЗ, є:

- помилки, приховані в самій програмі;
- спотворення вхідної інформації;
- невірні дії користувача;
- несправність апаратних засобів ИС, на якій реалізується обчислювальний процес.

Помилки, приховані в програмі. При розробці складного ПО можливе виникнення помилок, які не завжди вдається виявити і ліквідовувати в процесі відлагодження. В силу цього в програмах залишається деяка кількість прихованих помилок. Вони є причиною невірного функціонування цих програм. Серед помилок подібного роду можна виділити наступні характерні групи.

Помилки обчислень. Помилки цієї групи пов'язані з некоректним записом або програмуванням математичних виразів, а також невірне перетворення типів змінних. Внаслідок цього виходять неправильні результати.

Логічні помилки. Ця група помилок є причиною спотворення алгоритму рішення задачі. До помилок подібного роду можна віднести невірну передачу управління, невірне завдання діапазону зміни параметра циклу, невірну умову і інші помилки.

Помилки введення-виведення. Ці помилки пов’язані з неправильним управлінням введення-виведення, формуванням вихідних записів, визначенням розміру записів і іншими неправильно здійсненими діями.

Помилки маніпулювання даними. Таких помилок належать: невірне визначення числа елементів даних; невірні початкові значення, присвоєні даним; невірна вказівка довжини операнду або імені змінної і інші помилки.

Помилки сумісності пов'язані з відсутністю сумісності вживаного ПО, що розробляється або, з операційною системою або іншими застосовними програмами.

Помилки сполучень. група цих помилок викликає невірну взаємодію

ПЗ з іншими програмами або підпрограмами, з системними програмами, облаштуваннями ЕОМ або вхідними даними.

1.3 Висновки до першого розділу

В результаті аналізу сучасного стану питання щодо забезпечення якості програмного забезпечення стає очевидним, що галузь розробки ПЗ знаходиться в стані тривалої кризи. І це не дивно з огляду на те, що лише 30 % проектів по розробці програмного забезпечення взагалі доводяться до стадії завершеного продукту придатного до використання.

Очевидно, що існуючі підходи до контролю та управління якістю не здатні забезпечити галузь адекватним інструментарієм для того, щоб виправити ситуацію, що склалась. Адже різноманітні методи тестування та виявлення помилок на стадії випуску готового програмного продукту не гарантують, що знайдені помилки не будуть з’являтись в майбутніх проектах. До того ж існуючі підходи не скорочують час який витрачається на розробку ПЗ. Тому, все частіше спеціалісти з контролю якості звертають увагу на те, що якість є комплексним динамічним поняттям і підходи до її контролю мають змінюватись разом зі змінами в системі «продукт – споживач» в цілому.

А беручи до уваги, що процес розробки програмного забезпечення є багатостадійним, доходять висновку, що якість повинна контролюватись на всіх етапах розробки, і ступінь контролю повинен залежати від поточної стадії.

2 ВИЗНАЧЕННЯ МЕТРИК ЯКОСТІ НА РАННІХ ЕТАПАХ ЖИТТЄВОГО ЦИКЛУ ПЗ

Аналіз якості програмного забезпечення показує, що дуже велика кількість помилок в програмному забезпеченні будь якого рівня (від вбудованого до прикладного) були внесені на ранніх етапах життєвого циклу. Якщо процес розробки програмного забезпечення розбити на три етапи: формулювання вимог, проектування та конструювання, то можна помітити, що більшість помилок вноситься саме на етапі конструювання [4]. Розподіл помилок на різних стадіях розробки ПЗ зведений до таблиці 2.1.

Таблиця 2.1 – Розподіл помилок, які були допущені на різних стадіях розробки ПЗ

Етап життєвого циклу ПЗ	Об’єм програмного забезпечення				
	до 2КБ	до 8КБ	до 32КБ	до 128КБ	до 512КБ
Формулювання вимог	10%	15%	20%	22%	23%
Проектування	15%	19%	25%	28%	32%
Конструювання	75%	66%	55%	50%	45%

З таблиці 2.1 видно, що чим більший об’єм програмного забезпечення тим більше помилок вноситься саме на ранніх етапах створення програмного забезпечення. Слід відмітити і той факт, що вартість виправлення помилки тим більша чим більший за номером етап розробки. Таким чином вартість виправлення помилки, яка була внесена на стадії проектування в 2-4 рази вища за вартість виправлення помилки, що внесена на стадії конструювання. Залежність орієнтовної вартості виправлення програмної помилки від стадії проектування наведена на рисунку 2.1 [28].

Проектування ПЗ є формальним процесом який можна вивчати і вдосконалювати, отже створивши правильний підхід до створення програмного забезпечення можна суттєво підвищити його якість і забезпечити керованість процесу розробки і збільшити термін його існування.

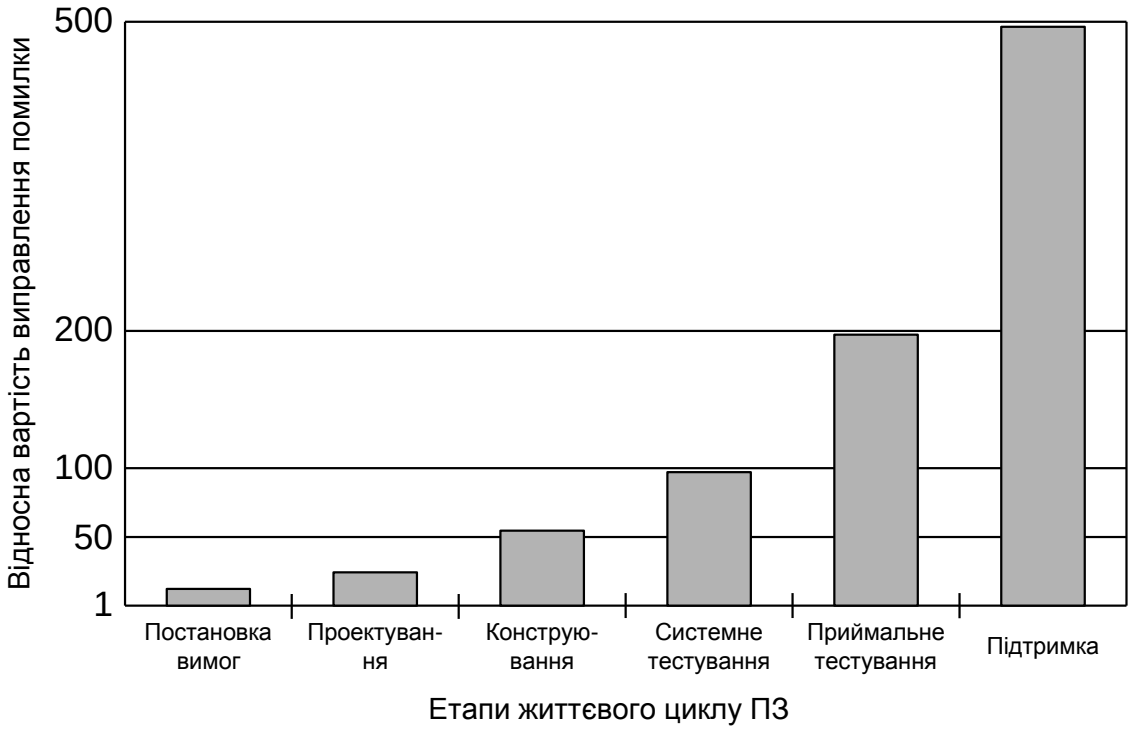

Рисунок 2.1 – Залежність орієнтовної вартості виправлення програмної помилки від стадії проектування

Отже для мінімізації витрат на розробку та підтримку програмного забезпечення та підвищення якості необхідно приділяти особливу увагу процесу контролю якості саме на ранніх етапах створення ПЗ. Для чого необхідно визначити метрики та визначитись з її діапазонами змін.

2.1 Метрики якості на ранніх етапах створення ПЗ

Метрики потрібні, щоб ефективно управляти процесом розробки ПЗ: діагностувати проблеми, відокремлювати їх, виправляти і перевіряти, чи правда обрані вами способи вирішення проблеми допомагають.

Кожного разу, впроваджуючи їх, будь-якій команді дуже некомфортно: доводиться зберігати додаткову інформацію, щось там міряти, розводити бюрократію. Але коли уперше від якої-небудь метрики отримується користь, на зміну лінії приходять дисципліна і глибоке розуміння важливості тієї або

іншої метрики. А якщо не приходять, це означає метрику можна сміливо відкинути.

Вирішуючи задачу створення множини метрик якості ПЗ на ранніх етапах життєвого циклу необхідно визначити що саме можна вважати раннім етапом життєвого циклу. Незалежно від того, яка модель життєвого циклу ПЗ [29] застосовується, очевидно, що до ранніх етапів слід віднести: постановку вимог, проектування та конструювання. Моделі життєвого циклу наведені в додатку Б.

Відносно кожного з перелічених етапів необхідно сформувати перелік метрик якості та визначитись з діапазоном зміни їх значень.

На ранніх етапах життєвого циклу ПЗ, зокрема на етапі розробки немає можливості використовувати показники якості (метрики), які отримані з використанням таких методів як: вимірювальний (оскільки немає готового продукту, немає чого вимірювати); реєстраційний (відсутнє середовище для реєстрації будь чого); органолептичний (без наявності готового продукту немає можливості використовувати органи чуття для аналізу якості). Тому на етапі проектування є можливість визначити якість програмного забезпечення лише з використанням розрахункового та експертного методу. А отже з використанням метрик розміру програми та метрик складності потоку управління та даних програми.

До групи оцінок розміру програм можна віднести також і метрику Холстеда. Основу метрики Холстеда складають чотири вимірювані характеристики програми: n1 – число унікальних операторів програми, включаючи символи-роздільники, імена процедур і знаки операцій (словник операторів); n2 – число унікальних операндів програми (словник операндів); N1 – загальне число операторів в програмі; N2 – загальне число операндів в програмі.

Спираючись на ці характеристики, що отримуються безпосередньо при аналізі початкових текстів програм, М. Холстед вводить наступні оцінки: словник програми

$$n1=n1+n2, \quad (2.1)$$

довжину програми

$$N=N1+N2, \quad (2.2)$$

об'єм програми

$$V=N*\log2\,(n)\,. \quad (2.3)$$

Мається на увазі, що програма складається з логічних одиниць інформації – символів, операторів, операндів. Вводиться поняття n* – теоретичний словник програми, тобто словниковий запас, необхідний для написання програми, з урахуванням того, що необхідна функція вже реалізована в цій мові а, отже, програма зводиться до виклику цієї функції.

Наприклад можливе здійснення процедури виділення простого числа могло б виглядати так:

CALL SIMPLE (X, Y),

де Y – масив чисельних значень, що містить шукане число X.

Теоретичний словник в цьому випадку складатиметься з

n1* :{CALL, SIMPLE (...)},

n1*=2; n2* :{X, Y},

n2*=2,

а його довжина, визначувана як n* = n1* + n2*, дорівнюватиме 4.

Використовуючи n*, вводять оцінку

$$V^*: V^* = n^* * \log2\, n^*, \quad (2.4)$$

за допомогою якої описується потенційний об'єм програми, що відповідає максимально компактному тексту програми, що реалізовує цей алгоритм.

Друга найбільш представницька група оцінок складності програм – метрики складності потоку управління програм. Як правило, за допомогою цих оцінок оперують або щільністю переходів, що управляють, усередині програм, або взаємозв'язками цих переходів. І в тому і в іншому випадку стало традиційним представлення програм у вигляді орієнтованого графа G= (V, E), що управляє, де V – вершини, що відповідають операторам, а E – шляхи, що відповідають переходам.

Уперше графічне представлення програм було запропоноване Маккейбом. Основною метрикою складності він пропонує визначати цикломатичну складність графа програми, або, як її ще називають, цикломатичне число Маккейба, що характеризує трудомісткість тестування програми. Для обчислення цикломатичного числа Маккейба Z (G) застосовується формула

$$Z(G) = e - v + 2p, \qquad (2.5)$$

де e – число дуг орієнтованого графа G;

v – число вершин;

p – число компонентів зв'язності графа.

Число компонентів зв’язності графа можна розглядати як кількість дуг, які необхідно додати для перетворення графа в сильно зв’язний. Сильно зв’язаним називається граф, будь-які дві вершини якого взаємно досяжні. Для графів коректних програм, т. е. графів, що не мають недосяжних від точки входу ділянок і «висячих» точок входу і виходу, сильно зв’язний граф, як правило, отримують шляхом замикання дугою вершини, що означає кінець програми, на вершину, що означає точку входу в цю програму.

По суті Z (G) визначає число лінійно незалежних контурів в сильно зв’язному графові. Інакше кажучи, цикломатичне число Маккейба показує необхідну кількість проходів для покриття усіх контурів сильно зв’язного графа або кількість тестових прогонів програми, необхідних для вичерпного тестування за критерієм «працює кожна гілка».

Цикломатичне число залежить тільки від кількості предикатів, складність яких при цьому не враховується. Наприклад, є два оператори умови:

```
if(x>0)
x=a;
else;
```

та

```
if((x>0 && FLAG==1) || (x==0) && FLAG==0)
x=a;
else;
```

Обидва оператори припускають однакове розгалуження і можуть бути представлені одним і тим же графом. Очевидно, цикломатичне число буде для обох операторів однаковим, не відбиваючим складності предикатів, що дуже істотно при оцінці програм.

Іще одна метрика складності програм, що дістала назву «Підрахунок точок перетину», авторами якої є М. Вудвард, М. Хенел і Д. Хидлей. Метрика орієнтована на аналіз програм, при створенні яких використовувалося неструктурне кодування на таких мовах, як асемблер і фортран. У графі програми, де кожному операторові відповідає вершина, т. е. не виключені лінійні ділянки, при передачі управління від вершини *a* до *b* номер оператора а рівний min (*a, b*), а номер оператора *b* – max(a, b). Точка перетину дуг з'являється, якщо

$$\min(a, b) < \min(p, q) < \max(a, b) \;\&\&\; \max(p, q) > \max(a, b) \;\|\; \|\; \min(a, b) < \max(p, q) < \max(a, b) \;\&\; \min(p, q) < \min(a, b).$$

Іншими словами, точка перетину дуг виникає у разі виходу управління за межі пари вершин (*a, b*). Кількість точок перетину дуг графа програми дає характеристику не структурованості програми.

Однією з найбільш простих, але, як показує практика, досить ефективних оцінок складності програм являється метрика Т. Джилба, в якій логічна складність програми визначається як насиченість програми виразами типу IF-THEN-ELSE. При цьому вводяться дві характеристики: CL – абсолютна складність програми, що характеризується кількістю операторів умови; cl – відносна складність програми, що характеризується насиченістю програми операторами умови, тобто. cl визначається як відношення CL до загального числа операторів. Використовуючи метрику Джилба, її

доповнюють ще однією складовою, а саме характеристикою максимального рівня вкладеності оператора CLI, що дозволяє не лише уточнити аналіз по операторах типу IF-THEN-ELSE, але і успішно застосувати метрику Джилба до аналізу циклічних конструкцій.

Великий інтерес викликає оцінка складності програм за методом граничних значень. Використовуючи цей метод вводиться декілька додаткових понять, пов'язаних з графом програми. Нехай G= (V, E) – орієнтований граф програми з єдиною початковою і єдиною кінцевою вершинами. У цьому графові число вершин, що входять, у дуг називається негативною мірою вершини, а число дуг, що виходять з вершини – позитивним степенем вершини. Тоді набір вершин графа можна розбити на дві групи: вершини, у яких позитивний степінь <=1; вершини, у яких позитивний степінь >=2. Вершини першої групи назвемо приймаючими вершинами, а вершини другої групи - вершинами відбору. Для отримання оцінки по методу граничних значень необхідно розбити граф G на максимальне число підграфів G', що задовольняють наступним умовам: вхід в підграф здійснюється тільки через вершину відбору; кожен підграф включає вершину (що називається надалі нижньою межею підграфа), в яку можна потрапити з будь-якої іншої вершини підграфу.

Число вершин, що утворюють такий підграф, дорівнює скоректованій складності вершини відбору. Кожна приймаюча вершина має скоректовану складність, рівну 1, окрім кінцевої вершини, скоректована складність якої дорівнює 0. Скоректовані складнощі усіх вершин графа G підсумовуються, утворюючи абсолютну граничну складність програми. Після цього визначається відносна гранична складність програми:

$$S_0=1-(v-1)/S_a, \qquad (2.6)$$

де S_0 – відносна гранична складність програми;

S_a – абсолютна гранична складність програми;

v – загальне число вершин графа програми.

Метрика, що зв'язує складність програм із зверненнями до глобальних змінних. Пара «модуль-глобальная змінна» позначається як (*p, r*), де *p* – модуль, що має доступ до глобальної змінної *r*. Залежно від наявності в програмі реального звернення до змінної *r* формуються два типи пар «модуль-глобальная змінна»: фактичні і можливі. Можливе звернення до *r* за допомогою *p* показує, що область існування *r* включає *p*. Характеристика A_{up} говорить про те, скільки разів модулі U_p дійсно діставали доступ до глобальних змінних, а число P_{up} – скільки разів вони могли б отримати доступ. Відношення числа фактичних звернень до можливих визначається

$$Rup = Aup/Pup. \quad (2.7)$$

Ця формула показує наближену вірогідність посилання довільного модуля на довільну глобальну змінну. Чим вище така ймовірність, тим вище ймовірність «несанкціонованої» зміни певної глобальної змінної, що призводить до ускладнення модифікації програми.

2.2 Діапазон значень метрик якості ПЗ

За визначенням стандарту [12] метрика якості ПЗ є «моделлю виміру атрибуту, що зв'язується з показником його якості». При вимірі показників якості ПО стандарт рекомендує використати наступні типи мір:

- міри розміру в різних одиницях виміру (кількість функцій, розмір програми, об'єм ресурсів та ін.);
- міри часу, періоди реального, процесорного або календарного часу (час функціонування системи, час виконання компонента, час використання та ін.);

- міри зусиль, продуктивний час, витрачений на реалізацію проекту (продуктивність праці окремих учасників проекту, колективна трудомісткість та ін.);
- міри інтервалів між подіями, наприклад час між послідовними відмовами;
- рахункові міри, лічильники для визначення кількості виявлених помилок, структурної складності програми, числа несумісних елементів, числа змін (наприклад, число виявлених відмов та ін.).

При оцінці значень показників якості залежно від особливостей використовуваних ними властивостей, способів їх визначення і призначення для кожної метрики якості застосовується певна шкала вимірів:

- шкала метрична (абсолютна, відносна, інтегральна);
- шкала порядкова (рангова), що дозволяє ранжирувати характеристики шляхом порівняння з опорними значеннями;
- класифікаційна шкала, що характеризує наявність або відсутність даної властивості у оцінюваного ПО.

Показники, які обчислюються за допомогою метричних шкал, називаються кількісними, а показники, визначувані за допомогою порядкових і класифікаційних шкал, - якісними.

Для розрахунку максимальних значень метрик якості ПЗ введемо певні обмеження на проекти, метрики якості яких підлягають аналізу. Розрахуємо максимальні значення метрик якості ПЗ при використанні наступних обмежень:

1) кожен модуль реально одержує доступ до глобальної змінної стільки разів, скільки може одержати такий доступ;

2) ПЗ містить не більше 10000 рядків вихідного коду;

3) тип проекту не обумовлено;

4) постановка задачі займає 20 % від загальної тривалості розроблення ПЗ і проектної вартості; проектування – 30 % загальної тривалості і проектної вартості; програмування – 20 % загальної тривалості і проектної

вартості; тестування, налагодження і перевірка якості – 30 % загальної тривалості проектної вартості;

5) 25 % часу проектування займає побудова та модифікація моделей;

6) максимальна кількість помилок моделей та прототипів одного модуля не повинна перевищувати 50;

7) ПЗ має не більше 50 модулів;

8) 50 % часу та вартості, відведених на тестування, налагодження і перевірку якості, займає перевірка якості ПЗ;

9) кількість зовнішніх входів кожного модуля – 30, кількість зовнішніх виходів кожного модуля – 30, кількість зовнішніх запитів до кожного модуля – 30;

10) кожен модуль має максимум 20 внутрішніх логічних файлів та використовує 20 зовнішніх логічних файлів.

З урахуванням введених обмежень та вище наведених формул розрахуємо діапазони значень для метрик якості ПЗ. Дані розрахунків приведені в таблиці 2.2.

Таблиця 2.2 – Значення інтервалів метрик якості ПЗ

Метрики з точними значеннями	Значення
Метрика звертання до глобальних змінних	0..1
Кількість виявлених помилок при інспектуванні	0..2000
Метрики з прогнозованими значеннями	
Цикломатична складність	10..100
Відносна гранична складність програми	0..1

Незважаючи на те, що метрики дозволяють контролювати процес розробки, робота з ними пов’язана з деякими труднощами. По-перше, усі відомі на сьогодні метрики коду недостатньо значимі і точні. Вони не здатні забезпечити отримання об'єктивної картини про стан програмної системи, а лише видають показники, які визначені за заданим алгоритмом. По-друге, процес вимірювання може бути штучно спотворений за рахунок того, що

співробітники «оптимізуватимуть» свій програмний код так, щоб метрики видавали кращі результати. Крім того, формальне використання метрик не враховує досвід співробітників, рівень компанії і може принести не лише користь, але і шкоду. Проте метрики є досить корисним інструментом в руках розробників і менеджерів проектів, що дозволяє виявити моменти відходу розробки на нижчий якісний рівень і розпізнати найбільш складні ділянки в системі. Визначення числових показників може дати нові відомості про продукт, що розробляється, і допомогти більш грамотно планувати витрати на його подальший розвиток.

У світі програмування складно оцінити якість написаного розробником коду, складно виявити його продуктивність. У зв'язку з цим, метрики можуть використовуватися як для непрямих оцінок праці розробника, так і для оцінок якості програмного продукту. Наприклад, спрощено, якщо в програмі, що виводить на екран "Hello, World"! 50 класів, то хтось з чимось перемудрував. З точки зору якості програмного забезпечення можуть мати цінність такі метрики як цикломатична складність, кількість прокоментованих рядків коду, число методів і атрибутів класу, а також покриття коду тестами. На їх обробці найпростіше скласти уявлення про якість продукту, що розробляється, тому нерідко їх використовують в системах безперервної інтеграції і доставки.

2.3 Висновки до другого розділу

Проведений аналіз показує, що основна частина помилок ПЗ виникають саме на ранніх етапах життєвого циклу програмного забезпечення, а саме на стадії проектування і особливо конструювання. В таких умовах формування системи якості вимагає підсилення контролю до формування вимог до проектування та конструювання ПЗ. Складність виконання процедуру контролю полягає в тому, що на ранніх етапах розробки ПЗ неможливо застосовувати більшість методів для визначення

показників якості. Тому для визначення показників якості ПЗ на ранніх етапах життєвого циклу були застосовані розрахунковий та експертний методи.

Як перспективний напрямок розвитку системи якості був обраний напрямок розвитку статичного коду для контролю якості ПЗ на ранніх етапах життєвого циклу. В рамках цього підходу були проаналізовані основні метрики та обрані найбільш ефективні до яких були віднесені метрика звертання до глобальних змінних, кількість виявлених помилок при інспектуванні, цикломатична складність, Відносна гранична складність програми. Розраховані значення інтервалів метрик якості ПЗ.

3 ОЦІНКА ЕФЕКТИВНОСТІ МЕТРИК ЯКОСТІ НА РАННІХ ЕТАПАХ ЖИТТЄВОГО ЦИКЛУ ПЗ

3.1 Оцінка ефективності метрик якості на ранніх етапах життєвого циклу ПЗ

Аналіз метрик визначених в попередньому розділі показує, що існують метрики з малим діапазоном значень і існують метрики з дуже великим діапазоном значень.

Опрацювання метрик якості [30] з отриманими в попередньому розділі діапазонами значень призведе до того, що при визначенні якості ПЗ в цілому впливовими виявляться метрики з більшим діапазоном. Так метрика «кількість виявлених помилок при інспектуванні» з діапазоном значень 0..2000

Для оцінки інформації, яку містять метрики якості ПЗ на ранніх етапах життєвого циклу ПЗ використаємо критерії оцінювання інформації [31]:

1) релевантність – наявність зв'язку з проблемою (відповідність інтересам та проблемі) та здатність інформації внести ясність у процес розуміння проблеми;

2) достовірність – на скільки представлень опис відповідає дійсності;

3) значущість – розуміння інформації, повнота висвітлення предмету інтересу, своєчасність інформації та її достатність для прийняття рішень, а також кількість інформації:

$$I = H(\text{апріорна}) - H(\text{апостеріорна}), \qquad (3.1)$$

де H(апріорна) – ентропія системи (показник незнання системи) до експерименту,

H(апостеріорна) – ентропія системи після експерименту.

Тоді, якщо $I > 0$, то в результаті експерименту одержано нову інформацію, показник незнання системи зменшився.

Щодо релевантності метрик якості ПЗ, то рівень релевантності дорівнює 1, оскільки саме метрики якості відображають уявлення про якість проекту та програмне забезпечення. Достовірність метрик якості ПЗ дорівнює 1, оскільки використовуються лише класичні метрики якості, описані в галузевій літературі.

Оскільки релевантність і достовірність метрик якості ПЗ є сталими величинами, то основним критерієм оцінювання інформації, який впливатиме на оцінку ефективності метрик якості ПЗ, є саме значущість інформації.

Для кожної зазначеної метрики сформуємо підсумковий критеріальний показник ефективності з використанням адитивного критерію.

Адитивний критерій A формується шляхом ділення на число показників ефекту n суми добутків часткових показників ефекту l_i на коефіцієнти значущості i-го показника g_i.

$$A = \left(\frac{1}{n}\right) \cdot \sum_{i=1}^{n} (l_i \cdot g_i). \quad (3.2)$$

Значущість, а відтак і ефективність, кожної метрики якості ПЗ етапу проектування залежить від 4-х показників ефекту ($n = 4$): розуміння інформації, повнота висвітлення предмету інтересу, своєчасність інформації та її достатність для прийняття рішень, кількість інформації. Кожен показник ефекту розглядатимемо для спрощення розрахунків як значення з інтервалу [0, 1], де 0 – мінімальне значення показника, 1 – максимальне значення. Значущості показників ефекту для кожної метрики рівні, причому

$$\sum_{i=1}^{n} g_i = 1,$$

а отже g_i=1/4=0,25. Для подальших розрахунків необхідно розрахувати часткові показники ефекту l_i для кожної метрики якості ПЗ а відповідно і ефективність кожного критерію якості.

Показник «розуміння інформації» l_1 дорівнює 1 для кожної з метрик, оскільки метрики якості розраховані вірно, з повним розумінням. Показник «повнота висвітлення предмету інтересу» l_2 залежить від повноти висвітлення метрикою інформації, що обробляється, функцій обробки інформації та переліку модулів, режимів роботи ПЗ. Показник «своєчасність інформації та її достатність для прийняття рішень» l_3 залежить від достатності інформації про оброблювану інформацію, функції обробки інформації та перелік модулів, а також про режими роботи ПЗ. Для розрахунку показника «кількість інформації» l_4 приймемо, що після опрацювання всіх метрик якості є повна інформація про якість проекту та програмного забезпечення, тому кількість інформації I, яку містять 4 означені метрики якості разом, згідно формули (3.1) дорівнює 1. Також будемо вважати, що показник незнання якості ПЗ зменшується в рівному ступені після опрацювання кожної метрики якості, тому кількість інформації кожної метрики якості

$$I_{\text{м}} = \frac{1}{4} = 0{,}25.$$

Значення ефективності кожної з означених метрик зведені до табл. 3.1.

Аналіз табличних даних показує, що відносно високу ефективність мають метрики: відносна гранична складність програми та метрика звертання до глобальних змінних. Менш ефективними залишаються інші метрики, проте очевидним залишається той факт, що при оцінці якості програмного забезпечення на ранніх етапах життєвого циклу необхідно враховувати і метрики з малим діапазоном значень, тому що ефективність їх достатньо висока. Тобто не врахування цих метрик призведе до зниження ефективності оцінки якості ПЗ в цілому.

Таблиця 3.1 – Значення ефективності метрик на ранніх етапах життєвого циклу

№	Метрика якості	l_1	l_2	l_3	l_4	A
1	Метрика звертання до глобальних змінних	1	0,33	0,667	0,25	0,141
2	Кількість виявлених помилок при інспектуванні	1	0,25	0,33	0,25	0,114
3	Цикломатична складність	1	0,25	0,45	0,25	0,09
4	Відносна гранична складність програми	1	1	0,333	0,25	0,161

3.2 Висновки до третього розділу

Проведений в розділі аналіз метрик якості ПЗ показав, що існують метрики з малими діапазонами зміни значень і метрики діапазони значень яких коливаються в широких межах. При врахуванні таких метрик для оцінки якості програмного продукту в цілому необхідно враховувати цей факт, а для кожної метрики в такому випадку необхідно визначати ефективність використання.

Для обраних метрик визначені критерії ефективності, які показали, що суттєво впливають на результат оцінки якості не лише метрики діапазони значень яких знаходяться в широких межах, а при врахуванні інших метрик необхідно враховувати і менш значущі показники.

4 ОЦІНКА РЕЗУЛЬТАТІВ

Оцінка результату програмних проектів – завдання складне. Часто якісь показники потрібні ще до початку робіт, щоб спланувати майбутні витрати, інші використовуються для визначення розміру винагороди за вже виконану працю. Наука не коштує на місці і пропонує усі нові підходи до рішення цієї задачі, проте досі немає надійної універсальної методики, що дає гарантований результат. Тому у більшості випадків доводиться випробувати різні методи.

4.1 Лінійний підхід

У найпростішому випадку визначити вартість розробки ПЗ можна виходячи з кількісної оцінки трудовитрат Т (у деяких одиницях, наприклад людино-місяцях або людино-годинах) і їх питомої вартості Ц: С = Т × Ц.

Ціна однієї одиниці трудовитрат для індустрії розробки ПЗ формується, в основному, виходячи із заробітної плати і пов'язаних з нею нарахувань. Як правило, інші складові мають набагато меншу питому вагу, і ними частенько можна нехтувати.

Що стосується самих трудовитрат, то їх достовірне обчислення у сфері інтелектуальної діяльності, до якої, поза сумнівом, відноситься розробка ПЗ, виконати досить складно. Простий підхід може бути грунтований на наступній лінійній формулі: Т = Р × П, де Р – розмір початкового коду ПЗ; П – тимчасова продуктивність.

Ця примітивна формула активно застосовується і до цього дня, хоча її неспроможність була встановлена досить давно. Мабуть, найвідомішою роботою, в якій критикується цей підхід, є та, що витримала більше двадцяти видань по-справжньому класична книга Фредеріка Брукса (Frederick Brooks) "Міфічний людино-місяць, або Як створюються програмні системи", що уперше побачила світ ще в 1977 р.

На думку Брукса «...наші методи оцінки помилково плутають досягнутий прогрес з витраченими зусиллями». В першу чергу, це твердження торкається способу, яким вимірюється результат, - на зорі програмування не було знайдено нічого кращого, ніж використання в цих цілях кількості рядків коду. Про абсурдність такого показника для оцінки програмного проекту сказано дуже багато, що, втім, зовсім не означає, що він не застосовується сьогодні.

Із зростанням майстерності програміст зазвичай робиться «лаконічнішим», тобто видаваний їм код для вирішення одних і тих же завдань стає все компактніше, а це означає, що його простіше і дешевше відлагоджувати і супроводжувати. Проте вищезгадана формула зовсім не стимулює цього процесу.

4.2 Оцінка з використанням емпіричних даних

Незважаючи на певні досягнення розглянутих методів при оцінці трудомісткості реалізації програмних проектів, досвідчений інженер скаже, що кращий спосіб упізнати довжину шляху – це пройти його. Дійсно, ніщо не вселяє в нас більше упевненості, ніж минулий досвід.

Про те, що трудомісткість і, відповідно, вартість програмного проекту нелінійно залежить від об'єму робіт, було відомо ще в 1970-х роках, коли з'явилися перші наукові публікації, підкріплені результатами серйозних досліджень.

Ймовірно, першою нелінійною моделлю, що використовує емпіричні дані і знайшла практичне застосування при оцінці вартості ПЗ, стала SLIM (Software Life - cycle Model), запропонована в 1978 м. Лоуренсом Патнамом (Lawrence Putnam). Згідно з нею трудомісткість обчислюється за наступною формулою:

$$T = \left(\frac{P}{C}\right) \cdot t_d^{-4}.$$

Розмір проекту P найчастіше обчислюється у кількості рядків коду, хоча відомі випадки успішного застосування моделі і для інших одиниць, наприклад функціональних точок. C – чинник середовища, деяка технологічна константа, що враховує, окрім рівня технологій, також і продуктивність персоналу, яка може розрізнятися від команд до команди. t_d – обмеження на термін постачання (вроках).

4.3 Автоматизовані програмні продукти по оцінці якості ПЗ

Ця категорія інструментів дозволяє розрахувати різні метрики проекту, передусім на основі початкового коду і UML- діаграм (в першу чергу прецедентів і класів), хоча в загальному випадку може бути використано практично все, що відноситься до проекту і піддається виміру (вимоги, специфікації, документація, активність роботи з системою контролю версій, проходження тестів і ін.).

Почесне місце займають засоби підрахунку числа рядків коду (метрики SLOC і її похідних) - їх абсолютна більшість, як серед комерційних, так і серед безкоштовних продуктів, у тому числі з відкритим кодом.

Locmetrics - дуже простий безкоштовний продукт з мінімалістським інтерфейсом. У числі підтримуваних мов - C/C++, C#, Java, SQL – можливе обчислення не лише метрики SLOC і її різновидів, але і цикломатичної складності. Відсутність документації не є перешкодою для використання програми, оскільки розібратися в інтерфейсі з двох кнопок і двох полів введення зовсім нескладно. Гірше, що немає навіть опису методики розрахунку метрик. До недоліків також слід віднести відсутність хоч би найпростіших засобів побудови звітності або експорту даних в один з популярних форматів.

USC Codecount - безкоштовний продукт з відкритими початковими кодами на мові ANSI C, розроблений Університетом Південної Каліфорнії (University of Southern California, USC) – тією ж організацією, в якій були створені COCOMO/COCOMO II. З цієї причини USC Codecount є офіційним інструментом для підрахунку метрики SLOC при використанні вказаних моделей. До числа підтримуваних мов входять C/C++, C#, Java, JavaScript, SQL, Perl, XML, в документації вказано, що методика розрахунку відповідає прийнятою SEI для моделей CMM/CMMI.

По суті, USC Codecount є цілим набором інструментів, розроблених по єдиних принципах, – для кожної мови програмування є окремий проект, і компільована консольна програма може бути використана тільки для нього. Незважаючи на видимі незручності, цю особливість слід вважати якраз перевагою, оскільки таким чином максимально враховуються властивості (характеристики) конкретної мови.

USC Codecount дає можливість обчислювати кількість логічних і фізичних SLOC, порожніх рядків, коментарів, директив компілятора, описів даних, виконуваних інструкцій по файлах проекту окремо і сумарно. Також надається статистика по числу входжень ключових слів різних категорій і співвідношення між різними значеннями.

SLOCCount – безкоштовний продукт, розроблений Дэвидом Вилером (David A. Wheeler), поставляється у вигляді початкових кодів на мові C за ліцензією GNU GPL. Орієнтований на UNIX- платформи, хоча можливе застосування і в Windows після відповідної адаптації.

Серед тих, що розглядаються це, мабуть, найбільш універсальний інструмент – до числа підтримуваних мов входять Ada, Assembler, awk, Bourne shell (включаючи похідні: bash, ksh, zsh, pdksh), C, C++, C#, C shell (включаючи tcsh), COBOL, Expect, Fortran (включаючи Fortran 90), Haskell, Java, lex (включаючи flex), LISP (включаючи Scheme), make- файли, Modula3, Objective - C, Pascal, Perl, PHP, Python, Ruby, sed, SQL, TCL, Yacc.

Продукт дозволяє розрахувати фізичну кількість SLOC за проектом (у

розрізі окремих використовуваних мов), має можливості обчислення трудомісткості і вартості проекту на основі моделі COCOMO (за застарілою першою версією і тільки по базовій моделі). Завдяки своїй універсальності цей продукт досить популярний, приміром, саме він використовується в перспективному пошуковому механізмі для розробників Krugle для підрахунку статистики по проектах.

Code Counter Pro - єдиний комерційний продукт, що потрапив в нашому огляді в цю категорію. Втім, він зовсім недоріг ($25 за одну ліцензію), зате на відміну від попередніх має розвинений графічний інтерфейс, що, безумовно, робить його використання зручнішим.

Підтримуються наступні мови програмування : Java, JSP, C/C++, VB, PHP, HTML, Delphi/Pascal, ASM, XML, COBOL, є можливість декларувати нові.

Незважаючи на те що програма добре справляється зі своїм завданням і навіть дозволяє будувати детальні звіти, чого не може запропонувати, скажімо, Locmetrics, вона поступається розглянутим відкритим аналогам по кількості обчислюваних показників (тільки число фізичних рядків коду, коментарів, порожніх рядків, а також сумарні значення).

4.4 Висновки до четвертого розділу

Аналіз програмного забезпечення щодо оцінки результатів перевірки якості показує, що на ринку існує велика кількість програмних продуктів, які реалізують різноманітні підходи. Кожен із розглянутих має свій переваги та недоліки, проте для забезпечення якості в цілому доцільно застосовувати більш ніж один програмний продукт, що дозволить не в останню чергу автоматизувати процес оцінки якості.

5 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ

5.1 Розробка програмного забезпечення

Отже, спочатку, для розуміння послідовності дій, які описують програму від початку до кінця, складаємо алгоритм у вигляді блок-схеми (додаток А) [7].

Далі, розробляємо програмний код (додаток Б).

Провівши огляд розробленого програмного коду та виявивши помилки складаємо звіт (таблиця 5.1):

Таблиця 5.1 – Оглядові помилки

Номер помилки	Назва модуля/функції	Опис помилки	Важливість помилки (висока, середня, низька)	Помилка виправлена Так/Ні
1	entry	Не вірно складена умова продовження виконання цикла	Середня	Так
2	regit	Умова не відповідає алгоритму	Середня	Ні
3	show per	Некоректно застосовується конструкція умовного переходу	Середня	Так
4	main	Кількість умовних розгалужень не відповідає завданню	Середня	Так

5.2 Тестування програмного забезпечення

Для кожного модуля будуємо графи та обчислюємо циклометричні числа [8].

Розробляємо тестові випадки для кожного графу та представляємо їх у вигляді таблиці 5.2.

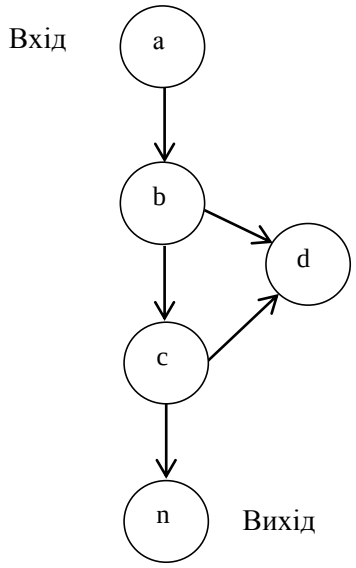

Рисунок 5.1– Граф модуля «entry»

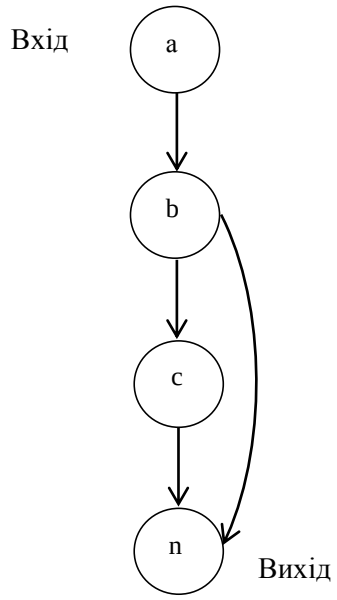

Рисунок 5.2 – Граф модуля «regit»

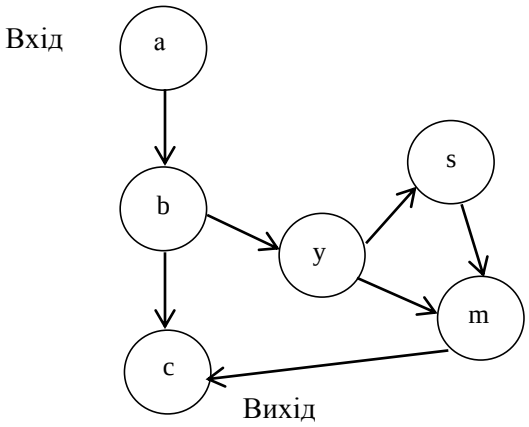

Рисунок 5.3 – Граф модуля «show per»

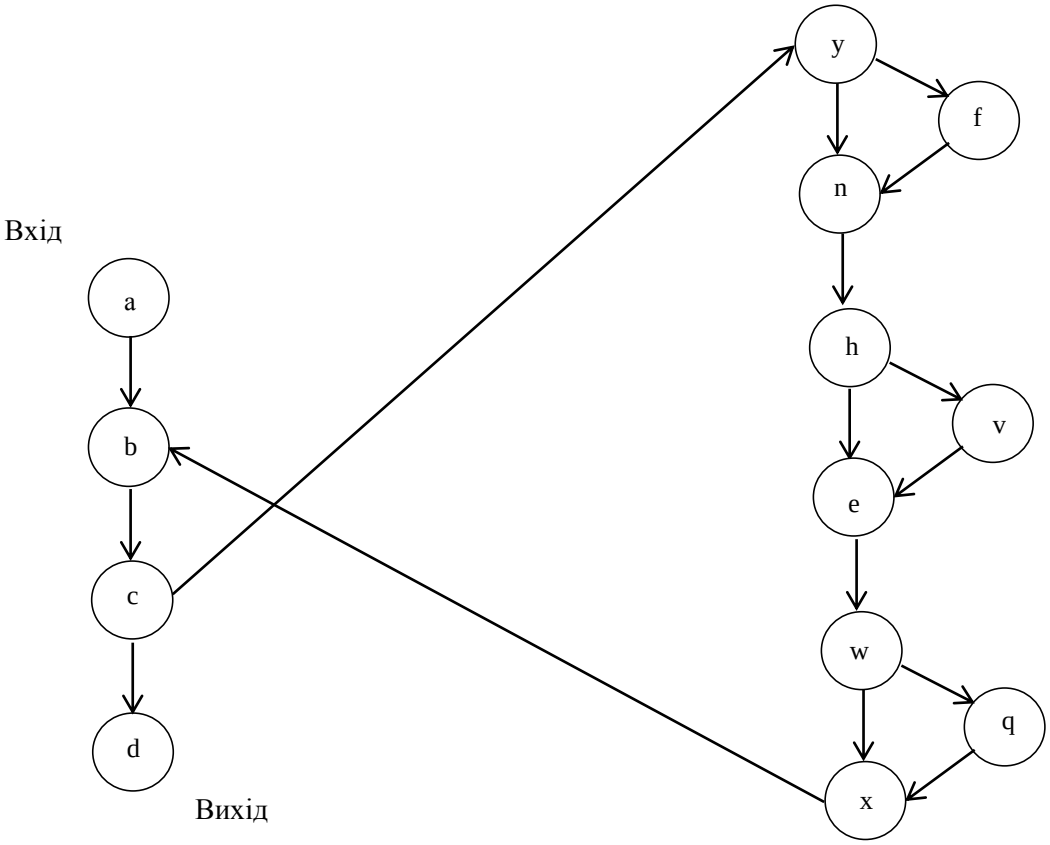

Рисунок 5.4 – Граф модуля «main»

Таблиця 5.2 – Тестові випадки

G	Номер сценарія	Опис проходу	Контрольні приклади, що дозволяють реалізувати описану ситуацію	Тест пройдений Так/Ні
Модуль entry				
2	1	a-b-c-d-b-c-n	a>0	Так
	2	a-b-c-n	a=0	Ні
Модуль regit				
2	1	a-b-n	fout.is_open=1	Так
	2	a-b-c-n	fout.is_open=0	Так
Модуль show per				
2	1	a-b-y-s-m-c	file.is_open=1, file.eof=1	Так
	2	a-b-c	file.is_open=0, file.eof=0	Так
Модуль main				
4	1	a-b-c-y-f-n-h-e-w-x-b-c-d	true=1, ans =1	Так
	2	a-b-c-y-f-n-h-y-e-w-x-b-c-d	true=1, ans =2	Так
	3	a-b-c-y-f-n-h-y-e-w-q-x-b-c-d	true=1, ans =3	Так
	4	a-b-c-d	true=0	Так

Обчислимо циклометричну складність для кожного модуля за формулою:

$$M = E - N + 2P, \qquad 5.1$$

де M – цикломатична складність;

E – кількість ребер в графі;

N – кількість вершин в графі;

P – кількість компонентів зв'язності.

Отже, отримаємо такі результати циклометричної складності для модулів «entry», «regit», «show per», та «main», відповідно:

$$M_{entry} = 5 - 5 + 2 \times 1 = 2.$$

$$M_{regit} = 4 - 4 + 2 \times 1 = 2.$$

$$M_{show\_per} = 7 - 6 + 2 \times 1 = 3.$$

$$M_{main} = 16 - 13 + 2 \times 1 = 5.$$

Далі проводимо модульне тестування згідно складеним тестовим випадкам [9]. Складаємо звіт про кожну знайдену похибку, таблиця 5.3.

Таблиця 5.3 – Модульне тестування

Номер помилки	Назва модуля	Опис проходу	Контрольні приклади, які дозволяють реалізувати описану ситуацію	Опис помилки	Важливість помилки	Помилка виправлена Так/Ні
1	entry	a-b-c-n	a=0	Некоректно відображаються введені дані	Середня	Так

Будуємо схему взаємодії модулів, рисунок 5.5.

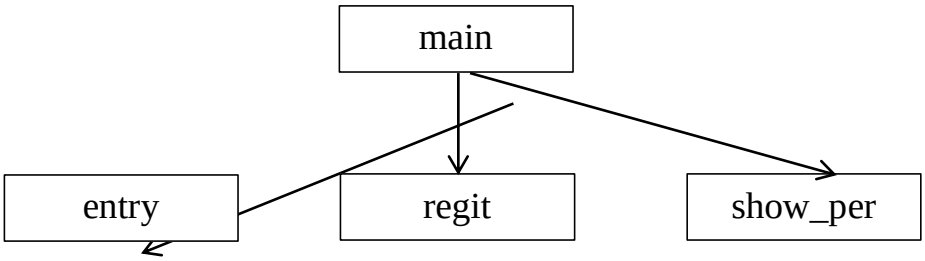

Рисунок 5.5 – Схема взаємодії модулів

Розробляємо стратегію тестування взаємодії модулів у вигляді таблиці 5.4 [10].

Таблиця 5.4 – Взаємодія модулів

Номер в послідовності	Опис послідовності	Контрольні приклади, які дозволяють реалізувати описану ситуацію	Тест пройдений Так/Ні
1	main→entry	true=1	Так
2	main→regit	ans=1	Так
3	main→show_per	ans=2	Так

5.3 Висновки до розділу

Проведено огляд розробленого програмного коду та складено звіт виявлених помилок.

Для кожного модуля побудовано графи та обчислено циклометричну скадність. Розроблено тестові випадки для кожного графу і представлено їх у вигляді таблиці.

Проведено модульне тестування згідно з складеними тестовими випадками, а також складено звіт про знайдені помилки. Побудовано схему взаємодії модулів.

Розроблено стратегію тестування взаємодії модулів.

6 ЕКОНОМІЧНА ЧАСТИНА

6.1 Оцінювання комерційного потенціалу розробки

Метою магістерської кваліфікаційної роботи є пошук шляхів вдосконалення системи управління якості програмного забезпечення в рамках існуючих методик та підходів до оцінки якості. Для визначення комерційного потенціалу розробки проведемо технологічний аудит.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів Вінницького національного технічного університету, кафедри метрології та промислової автоматики: Маньковська В.С., Кучерук В.Ю., Кулаков П.І. За допомогою таблиці 6.1 за п'ятибальною шкалою використовуючи 12 критеріїв оцінки комерційного потенціалу розробки експерти надали свої оцінки.

Таблиця 6.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Кри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів

Продовження табл. 6.1

5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання вел. к-сті дозвільних документів на в-во та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки пові-домлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 6.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 6.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 6.3 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Маньковська В. С.	Кучерук В.Ю.	Кулаков П. І.
	Бали, виставлені експертами:		
1	4	4	3
2	3	2	2
3	4	4	3
4	4	4	3
5	2	3	4
6	3	2	3
7	4	3	3
8	2	3	3
9	3	4	4
10	2	4	2
11	3	3	2
12	4	3	4
Сума балів	$СБ_1$=36	$СБ_2$=37	$СБ_3$=36
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_1^3 СБ_i}{3} = \frac{36+37+36}{3} = 36$		

Середньоарифметична сума балів, розрахована на основі висновків експертів склала 36, що згідно таблиці 6.2 вважається, що рівень комерційного потенціалу розробки є вище середнього.

Система управління якості програмного забезпечення призначена для застосування в процесі життєвого циклу будь-якого програмного продукту для підтримання належного рівня якості і як наслідок технічної та економічної доцільності програмного забезпечення. Призначена для застосування в IT галузі, підприємствах малого та середнього рівня технічної автоматизації.

В якості аналога для розробки було обрано 4 стандарти: Software engineering – Product quality, ISO 9126.

Основними недоліками аналога є те, що метрики програмного забезпечення не ранжируються по стадіям створенням проекту

У розробці дана проблема вирішується збільшенням ефективності визначення кількісних характеристик метрик, що застосовуються на ранніх етапах розробки.

6.2 Прогнозування витрат на виконання науково-дослідної роботи

1. Основна заробітна плата – винагорода за виконану роботу відповідно до встановлених норм праці. Вона встановлюється у вигляді тарифних ставок (окладів) і відрядних розцінок для робітників та посадових окладів для службовців. Стаття «Основна заробітна плата робітників» містить витрати на виплату основної заробітної плати робітникам, зайнятим виробництвом продукції.

Основна заробітна плата кожного із розробників (дослідників) *З* розраховується за формулою:

$$З = \frac{M}{T_P} \cdot t \ ,[грн.] \ , \qquad (6.1)$$

де M – місячний посадовий оклад конкретного розробника.

T_P – число робочих днів, $T_р = 22$;

t – число днів роботи розробника.

Розрахунки основної заробітної плати зведемо в таблицю 6.4:

Таблиця 6.4 – Розрахунок основної заробітної плати розробників

Найменування посади	Місячний посадовий оклад,грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	9600	436,4	5	2182
Програміст	7000	318,2	35	11136
Всього				13318

2. До статті «Додаткова заробітна плата» відносяться витрати на виплату виробничому персоналу підприємства додаткової заробітної плати за працю понад установлені норми, заохочувальні виплати за поточну виробничу діяльність, компенсаційні виплати тощо. Звичайно, ці витрати встановлюються у відсотках до основної заробітної плати на підставі відповідних розрахунків на підприємстві:

$$З_д = 11\% \cdot З_\Sigma, \tag{6.2}$$

$$З_д = 11\% \cdot (13318) = 1465 (грн.).$$

3. Витати на соціальні заходи виникають внаслідок здійснення обов’язкової сплати єдиного внеску на загальнообов’язкове державне соціальне страхування. Відрахування на соціальні заходи здійснюється від суми всіх витрат на оплату праці робітників, зайнятих безпосередньо виробництвом продукції:

$$В_{сз} = (З_\Sigma + З_д) \cdot \frac{\beta}{100\%}, \tag{6.3}$$

де в – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, %.

З 1.01.2016 року ставка єдиного внеску на загальнообов'язкове державне соціальне страхування встановлена залежно від класу професійного ризику виробництва і для бюджетної сфери в=22,0% .

$$В_{сз}=(13318+1465\ )\cdot\frac{22,0\%}{100\%}=3252,3(грн.).$$

4. У спрощеному вигляді амортизаційні відрахування у загальному можуть бути розраховані за формулою:

$$A=\frac{Ц\cdot T}{Tкор\cdot 12}\ \ [грн], \tag{6.4}$$

де Ц – балансова вартість даного виду обладнання (приміщень), грн.;

$Т_{кор}$ – час користування;

Т – термін використання обладнання (приміщень), цілі місяці.

Згідно пункта 137.3.3 Податкового кодекса амортизація нараховується на основні засоби вартістю понад 2500 грн.

Всі проведені розрахунки амортизаційних відрахувань заносимо в табл. 6.5.

Таблиця 6.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень	Балансова вартість, грн.	$t_{кор}$ (р)	Термін використання міс.	Величина амортизаційних відрахувань, грн.
1.Комп’ютер	12000	2	1	500
2.Принтер	4000	2	1	166,67
Всього				666,67

5. Норма витрат матеріалу – це плановий показник, який визначає максимально допустимі затрати відповідних ресурсів на виробництво одиниці продукції в умовах певного рівня техніки і організації виробництва.

Витрати на матеріали М, що були використані під час виконання даного етапу роботи, розраховуються по кожному виду матеріалів за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \text{ грн.,} \quad (6.5)$$

де $Н_i$ – витрати матеріалу *і*-го найменування, кг;

$Ц_i$ – вартість матеріалу *і*-го найменування, грн./кг.;

$К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15);

$В_i$ – маса відходів матеріалу *і*-го найменування, кг;

$Ц_в$ – ціна відходів матеріалу *і*-го найменування, грн/кг;

n – кількість видів матеріалів.

Інформацію про використані матеріали подамо у вигляді табл. 6.6.

Таблиця 6.6 – Матеріали, що використані на розробку

Найменування матеріалу	Ціна за одиницю, грн.	Витрачено	Вартість витраченого матеріалу, грн.
Бумага	85	1	85
Ручка	10	1	10
Флешка	120	1	120
Тонер	400	0.05	20
Всього			235
З врахуванням коефіцієнта транспортування			258,5

6. До статті «Паливо та енергія на технологічні цілі» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються у процесі виробництва продукції. У даному випадку будемо враховувати лише витрати на електроенергію. Витрати на енергію визначаються на основі витрат на одиницю продукції та тарифів на енергію за допомогою залежності:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (6.6)$$

де В – вартість 1 кВт енергії, грн. В = 8,44 грн/кВт*год;

П – установлена потужність обладнання, кВт. При паяні використовується паяльник потужність П = 500 Вт або П = 0,5 кВт;

Ф – фактична кількість годин роботи обладнання, год. Ф = 130 год;

$К_п$ – коефіцієнт використання потужності, $К_п$ = 0,65.

$$В_е = 8,44 \cdot 0,5 \cdot 130 \cdot 0,65 = 356,59 (грн).$$

7. Інші витрати $B_{ін}$ охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо.

Інші витрати $B_{ін}$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$B_{ін} = (1..3) \cdot (З + З_р). \tag{6.7}$$

$$B_{ін} = 1 \cdot (13318) = 13318 (грн.)$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

$$В = 13318+1465+3252,3+666,67+258,5+356,59+13318=32635,4 \text{ грн.}$$

Загальна вартість всієї МКНР визначається за формулою:

$$B_{заг} = \frac{B}{\alpha} \tag{6.8}$$

$$B_{заг} = \frac{32635,4}{1} = 32635,4 (грн.)$$

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\beta},$$

(6.9)

де $\beta -$ коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії розробки дослідного зразка, то коефіцієнт β = 0,7.

Звідси:

$$ЗВ = \frac{32635,4}{0,7} = 46622\,(грн.).$$

6.3 Прогнозування комерційних ефектів від реалізації результатів розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $ДП_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_1^n \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \qquad (6.10)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

Цо – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2019 році – 18%.

Планується, що після запровадження нашої методики зросте продуктивність програмного забезпечення.

Припустимо, що при впроваджені результатів нової методики покращується якість, що дозволяє підвищити ціну його реалізації на 300 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 200 шт., протягом другого року – на 250 шт., протягом третього року на 280 шт. Реалізація продукції до впровадження розробки складала 1 т, а її ціна 1500 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = \left[300 \cdot 1 + (1500 + 300) \cdot 200\right] \cdot 0,833 \cdot 0,25 \cdot \left(1 + \frac{18}{100}\right) = 61548,8(\text{грн.})$$

$$\Delta П_2 = \left[300 \cdot 1 + (1500 + 300) \cdot (200 + 250)\right] \cdot 0,833 \cdot 0,25 \cdot \left(1 + \frac{18}{100}\right) = 138669\,(\text{грн.})$$

$$\Delta П_3 = \left[300 \cdot 1 + (1500 + 300) \cdot (200 + 250 + 280)\right] \cdot 0,833 \cdot 0,25 \cdot \left(1 + \frac{18}{100}\right) = 224766(\text{грн.})$$

6.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності. Теперішню вартість інвестицій PV, що вкладаються в наукову розробку приймемо рівну загальним витратам PV = ЗВ = 46622 грн.

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \tag{6.11}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{6.12}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2; t – період часу (в роках).

$$ПП = \frac{61548,8}{(1+0,2)^1} + \frac{138669}{(1+0,2)^2} + \frac{224766}{(1+0,2)^3} = 278266,82\,(грн.)\,.$$

$$\text{Е}_{абс} = (278266,82 \;\; - 46622\,) = 231644,79(грн.).$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{6.13}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{1 + \frac{231644,79}{46622}} - 1 = 0,81 = 81\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{6.14}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2018 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0,18 + 0,05 = 0,23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{6.15}$$

$$T_{ок} = \frac{1}{0,81} = 1,2 \text{ (роки)}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

6.5 Висновки до економічного розділу

В даному розділі було оцінено економічний потенціал вдосконалення системи управління якості програмного забезпечення, який виявився на вище середньому рівні.

Прогнозування витрат на виконання науково-дослідної роботи по кожній з статей витрат складе 32635,5 грн. Загальна ж величина витрат на виконання та впровадження результатів даної НДР буде складати 46622 грн.

Вкладені інвестиції в даний проект окупляться через 1,2 роки при прогнозованому прибутку 278266,82 грн. за три роки.

ВИСНОВКИ

Аналіз проведений в роботі вказує на те, що сьогодні існує велика кількість стандартів та технологій покликаних підвищувати якість програмного забезпечення, проте якість готової продукції як і раніше залишається на достатньо низькому рівні. Достатньо лише того, що лише 16 % програмних проектів розпочатих в попередніх роках завершились з дотриманням графіку та перейшли у фазу завершеного програмного продукту.

Очевидно, що існуючі підходи при всій своїй повноті та рівні автоматизації не здатні забезпечити достатній рівень якості ПЗ. Аналіз показав, що не малу частину контролю для забезпечення рівня якості необхідно проводити на ранніх етапах життєвого циклу програмного забезпечення.

В результаті дослідження були визначені метрики якості, та визначена їх ефективність. Обраховані граничні значенні та діапазони зміни спираючись на дані проведених досліджень. Серед найбільш ефективних метрик якості придатних до застосування на ранніх етапах життєвого циклу ПЗ були обрані: метрика звертання до глобальних змінних, кількість виявлених помилок при інспектуванні, цикломатична складність, відносна гранична складність програми. Результати досліджень показують, що на рівень якості впливають не тільки метрики з широким діапазоном зміни значень і нехтувати менш значущими метриками не доцільно.

Як перспективний напрямок розвитку системи якості був обраний напрямок розвитку статичного коду для контролю якості ПЗ на ранніх етапах життєвого циклу.

Була проведена оцінка результатів, в ході якої були проаналізовані сучасні програмні продукти, покликані автоматизувати процес визначення якості ПЗ на ранніх етапах життєвого циклу. В результаті аналізу сформовані рекомендації щодо застосування розглянутих інструментів.

Ефективність цикломатичної складності піднімається за рахунок попередньої кількісної оцінки алгоритмічних структур програми, які відрізняються від елементарних типу розгалуження, циклів на поряд, а той два і застосування цих заздалегідь отриманих даних для оцінки більш об’ємних структур.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Игорь Осколов. Intel iStep 2015: сколько в мире программистов? [Електронний ресурс]. – 2015. – Режим доступу: https://3dnews.ru/912876 (дата звернення: 15.05.2018)
2. Говорущенко, Т. О. Аналіз галузі оцінювання якості програмного забезпечення. [Електронний ресурс]. – 2013. – Режим доступу: http://ena.lp.edu.ua/ bistream/ntb/26404/1/9_41-48.pdf (дата звернення: 15.05.2018)
3. Максим Ищенко. Рынок труда 2016: 100 тыс. программистов, бум ИТ-школ, избыток PM. [Електронний ресурс]. – 2016. – Режим доступу: https://dou.ua /lenta/articles/jobs-and-trends-2016/ (дата звернення: 15.05.2018)
4. Поморова, О. В. Сучасні проблеми оцінювання якості програмного забезпечення / О. В. Поморова, Т. О. Говорущенко // Радіоелектронні і комп'ютерні системи. – 2013. – № 5. – С. 319-327.
5. The Standish Group Report CHAOS [Електронний ресурс]. – 2016. – Режим доступу: https://www.projectsmart.co.uk/white-papers/chaos-report.pdf (дата звернення: 15.05.2018)
6. Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. – М. : Издательство «Русская редакция», 2010. – 89 с. : ил.
7. Software quality – capability of software product to satisfy stated and implied needs when used under specified conditions: ISO/IEC 25000:2014(en) Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE
8. Оценка программной продукции. Характеристики качества и руководства по их применению. ГОСТ Р ИСО/МЭК 9126-93
9. The degree to which a system, component, or process meets customer or user needs or expectations: IEEE Std 610.12-1990. IEEE Standard Glossary of Software Engineering TerminologyISO/IEC 2382-1:1993, Information

technology – Vocabulary – Part 1: Fundamental terms.

10. ISO/IEC 9126-1:2001, Software engineering – Product quality – Part 1: Quality model.
11. ISO/IEC TR 9126-2:2003, Software engineering - Product quality - Part 2: External metrics.
12. ISO/IEC TR 9126-3:2003, Software engineering - Product quality - Part 3: Internal metrics.
13. ISO/IEC TR 9126-4:2004, Software engineering - Product quality - Part 4: Quality in use metrics.
14. Данильчук, А. А., Юн С. Г., Новокрещенов, Н. С. Характеристики и атрибуты качества систем мониторинга ИТ-инфраструктуры по ISO 9126 // Наука вчера, сегодня, завтра: сб. ст. по матер. VII междунар. науч.-практ. конф. № 7(7). – Новосибирск: СибАК, 2013.
15. Зручність проведення всіх видів діяльності, пов'язаних з супровід програм. [Електронний ресурс]. – 2015. – Режим доступу: https://studopedia.ru/7_14 4480_zruchnist-suprovodu-maintainability.html (дата звернення: 17.05.2018)
16. Матеріал з Вікіпедії – вільної енциклопедії. Метрика програмного забезпечення. [Електронний ресурс]. – 2018. – Режим доступу: https://uk. wikipedia.org/wiki/Метрика_програмного_забезпечення (дата звернення: 17.05.2018)
17. Галузева система управління якістю. Методи оцінки показників якості програмного забезпечення програмно-технічних комплексів критичного призначення. [Текст]: Настанова Національного космічного агентства України СОУ-Н НКАУ 0031:2007 / Конорев Б. М. (наук. керівник розробки). – 2007. – 127 с.
18. Дишлевий, О. П. Інспекція програмного забезпечення. Лекція 7. [Електрон-ний ресурс]. – 2010. – Режим доступу: https://ppt-

online.org/186731 (дата звернення: 23.05.2018)

19. Что такое метод Фагана и зачем он нужен? [Електронний ресурс]. – 2010. – Режим доступу: http://ru.qatestlab.com/knowledge-center/qa-testing-materi
als/what-is-fagan-inspection-and-why-do-you-need-it/ (дата звернення: 30.05. 2018)
20. Матеріал з Вікіпедії – вільної енциклопедії. Тестування програмного забезпечення. [Електронний ресурс] – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/Тестування_програмного_забезпечення (дата звернення: 05.06.2018)
21. Матеріал з Вікіпедії – вільної енциклопедії. Статический анализ кода. [Електронний ресурс] – 2018. – Режим доступу: https://ru.wikipedia. org/wiki/Статический_анализ_кода (дата звернення: 05.06.2018)
22. Михаил Моисеев. Методы анализа и обеспечения качества ПО. Введение в статический анализ. Курс лекций. [Електронний ресурс] – 2011. – Режим доступу: http://kspt.icc.spbstu.ru/media/ files/2012/course/ quality/lec2.pdf (дата звернення: 05.06.2018)
23. Матеріал з Вікіпедії – вільної енциклопедії. Обфускация (программное обеспечение). [Електронний ресурс] – 2018. – Режим доступу: https://ru. wikipedia.org/wiki/Обфускация_(программное_обеспечение) (дата звернення: 06.06.2018)
24. Майерс, Г. Надежность програмного обеспечения / Г. Майерс. – М .: Мир, 1980. – 359 с.
25. Гласс, Р. Руководство по надежному программированию / Р. Гласс. – М.: Финансі и статистика, 1982. – 256 с.
26. Феллер, В. Введение в теорию вероятностей и ее приложения. / Т.1. М.: – Мир, 1967. – 752 с.
27. John Fodeh. Making the financial case for testing. [Електронний ресурс] – 2018. – Режи доступу: https://www.slideshare.net/EuroSTARConference/ john-fodeh-spend-wisely-test-well-workshop-revised-john-fodeh (дата

звернення: 08.06.2018)

28. Матеріал з Вікіпедії – вільної енциклопедії. Життєвий цикл програмного забезпечення. [Електронний ресурс] – 2018. – Режим доступу: https://uk.wikipedia.org/wiki/Життєвий_цикл_програмного збезпечення (дата звернення: 08.06.2018)
29. Говорущенко, Т. О. Визначення ефективності метрик якості на етапі проектування програмного забезпечення. / Т. О. говорущенко, Є. В. Питлик. // Вісник Хмельницького національного університету. Технічні науки. №2, 2012, - С. 149-155
30. Нежданов, И. Ю. Аналитическая разведка для бизнеса. / И. Ю. Нежданов. – М.: – Изд-во «Ось-89», 2008, – 336 с.

Додатки