Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

ДИПЛОМНА РОБОТА

Удосконалення методу управління якості розробки програмних продуктів
Назва теми

Рівень вищої освіти Другий (магістерський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121 «Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр ДРІПЗ. 160118.01.07.ПЗ

Виконав студент 2 курсу, група ІПЗм-20-1 _____ О. Р. Мартинюк
Підпис Ініціали, прізвище

Керівник док. техн. наук, проф _____ О. В. Бармак
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер канд. техн. наук, доцент _____ О. М. Яшина
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

3 грудня 2021 р.

Хмельницький 2021

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Другий (магістерський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
01 09 2021 р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ)

Мартинюку Олександру Руслановичу
Прізвище, ім'я, по батькові студента

1. Тема проєкту (роботи) Удосконалення методу управління якістю розробки програмних продуктів

Керівник проєкту (роботи) Бармак Олександр Володимирович
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 25.08.2021 р. № 102
2. Строк подання студентом проєкту (роботи) на кафедру 01.12.2021 р.
3. Вихідні дані до проєкту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
1 Аналіз сучасних методів управління якістю розробки програмних продуктів
2 Модель та методи системи управління якістю розробки програмних продуктів
3 Архітектура програмної реалізації системи управління якістю розробки програмних продуктів
4 Програмна реалізація системи управління якістю розробки програмних продуктів
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)
Презентаційні матеріали (слайди)

6. Консультанти розділів дипломного проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання « 01 » вересня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1. Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження; визначення структури дипломної роботи	01.09 - 07.09.2021	
2. Аналіз предметної області, актуальних технологій, моделей та методів	08.09 - 25.09.2021	
3 Розробка моделей та методів вирішення завдання	26.09 - 10.10.2021	
4. Робота над науковими публікаціями	11.10 - 20.10.2021	
5. Проектування архітектури системи для вирішення задачі, розробка вимог.	11.10 - 26.10.2021	
6 Детальний опис реалізації системи та тестування	27.10 - 15.11.2021	
7 Оформлення пояснювальної записки згідно вимог чинних стандартів	16.11 - 30.11.2021	
8 Попередній захист дипломної роботи	17.11.2021	
9 Перевірка роботи на наявність плагіату; норм контроль; брошурування пояснювальної записки; підготовка супровідних документів	01.12 - 04.12.2021	
10 Підготовка до захисту дипломної роботи	05.12 - 08.12.2021	

Студент _____ О. В. Мартинюк
Підпис Ініціали, прізвище

Керівник проєкту (роботи) _____ О. В. Бармак
Підпис Ініціали, прізвище

РЕФЕРАТ

Тема дипломної роботи: «Удосконалення методу управління якістю розробки програмних продуктів».

Автор роботи: Мартинюк Олександр Русланович.

Керівник роботи: Бармак Олександр Володимирович.

Пояснювальна записка: 82 с., 24 рис., 3 дод., 33 джерел.

СИСТЕМА УПРАВЛІННЯ ЯКІСТЮ, МЕТРИКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, ЗАСОБИ КОНТРОЛЮ ЯКОСТІ, ПРОГРАМНИЙ ПРОДУКТ.

Об'єктом дослідження є процеси життєвого циклу програмного забезпечення.

Предметом дослідження є методи та підходи управління якістю розробки програмних продуктів.

Метою дослідження є удосконалення методу управління якістю розробки програмних продуктів.

У роботі використані наступні методи дослідження та апаратура:

- аналіз, синтез, абстрагування, порівняння, опис та тестування;
- програмні компоненти Visual studio 2019 та MindFusion.WPF;
- персональний комп'ютер.

У кваліфікаційній роботі було виконано дослідження існуючих рішень з метою проведення порівняльного аналізу моделей розробки програмного забезпечення, порівняння систем менеджменту якості та визначення методів покращення якості коду та процесів розробки. Обгрунтовано доцільність розробки системи менеджменту якості розробки програмних продуктів, яка усуватиме недоліки моделей розробки програмного забезпечення. Було сформовано структурні компоненти системи управління якістю розробки програмного забезпечення, виділено основні її принципи та регламентовано використання методів контролю якості коду та методів контролю якості процесів. Виконане проектування програмної реалізації системи менеджменту якості, що включає в себе побудову інформаційної структури розроблюваної системи, визначення

парадигми програмування та побудову детального проекту системи. Здійснено вибір мови програмування, засобів та інструментів розробки, за допомогою чого було розроблено та детально описано програмну реалізацію системи управління якості програмних продуктів, проведено її тестування та аналіз.

Наукова новизна дослідження:

– отримав подальший розвиток підхід загального управління якістю у напрямку його застосування у сфері розробки програмних продуктів, що дозволяє підвищити якість програмного коду та процесів його розробки;

– удосконалено систему менеджменту якості TQM у сфері розробки програмних продуктів, що поєднує як засоби аналізу якості коду, так і методи статистичного аналізу якості процесів;

– удосконалено метод управління якості розробки програмних продуктів.

Практичне значення отриманих результатів полягає у вдосконаленні методу управління якістю розробки програмних продуктів, шляхом впровадження отриманої системи менеджменту якості у процес розробки, що дає змогу суттєво покращити ефективність роботи команди шляхом підвищення якості коду та оптимізації організаційних процесів.

У подальших дослідженнях даний метод можна удосконалювати а допомогою інтеграції нових засобів контролю якості, нових метрик, типів тестувань та методів статистичного аналізу.

22.11.2021 _____

ABSTRACT

Master's thesis: «Improving the method of software development quality management».

Author: Martyniuk Oleksandr Ruslanovych.

Head of work: Barmak Oleksandr Volodymyrovych.

Master's thesis consists of: 82 pages of the general text, 24 graphics, 3 supplements, 33 literature sources.

QUALITY MANAGEMENT SYSTEM, SOFTWARE METRIC, QUALITY CONTROL TOOLS, SOFTWARE PRODUCT.

The object of the study is the software development life cycle and its processes.

The research focuses on the methods and approaches of software development quality management.

The aim of the investigation is to develop the improved method of software development quality management by introducing a quality management system.

The following methods were conducted in the course of the research:

- analysis, synthesis, abstraction, comparison, description and testing;
- software components Visual Studio 2019 and MindFusion.WPF;
- a personal computer.

The qualification work contains an investigation of the existing solutions with the purpose of conducting a comparing analysis of software development models, a comparison of quality management system and defining the ways of improving software code quality and development processes. The relevance of developing a software quality management system, that will avoid the models of software development flaws. was justified. Structural components of the software development quality management system were defined, its principles were outlined, the code quality control tools and processes quality control tools were determined. A software implementation of the quality management system was designed, it included a specification of the software informational structure, a choosing of the programming paradigm and a detailed design definition. The programming language was chosen, instruments and tools of the

development were chosen. They were used to develop and create a detailed description of the software implementation of the software development quality management system, its testing and analysis were conducted.

The scientific novelty of the research lies in the following statements:

- the total quality management approach has got an advancement in the field of its application in the area of software development, what allows to rise the quality of software and its development processes;
- a software quality management system was developed, it combines tools for code quality analysis and tools for processes statistical analysis;
- the method of software development quality management was improved by integrating a quality management system that implements TQM principles and corresponds to the statements of ISO 9000 series.

The practical value of the obtained results lies in the improvement of software development quality control method by integrating a quality management system in the development process, which gives an ability to substantially improve the effectiveness of the teamwork by improving the quality of the code and the organization processes.

The study examined the existing methods of modeling the behavior of game artificial intelligence, their main advantages and disadvantages are investigated, and the requirements for their improvement are formed.

In the further researches this method can be enhanced by integrating new tools of quality control, new software metrics, new software testing types and statistical analysis methods.

22.11.2021

ЗМІСТ

Вступ 9
1 Аналіз сучасних методів управління якістю розробки програмних продуктів 12
1.1 Аналіз сучасних моделей розробки програмного забезпечення 12
1.2 Аналіз систем менеджменту якості 19
1.3 Аналіз засобів контролю якості 25
1.4 Постановка задачі 28
1.5 Висновки 30
2 Модель та методи системи управління якістю розробки програмних продуктів 31
2.1 Компоненти системи управління якістю розробки програмних продуктів 31
2.2 Засоби контролю якості програмного забезпечення 35
2.3 Засоби контролю якості процесів розробки 39
2.4 Висновки 41
3 Архітектура програмної реалізації системи управління якістю розробки програмних продуктів 43
3.1 Формування та аналіз вимог програмної реалізації системи управління якістю 43
3.2 Розробка архітектури програмної реалізації системи управління якістю . 45
3.3 Проєктування модулів системи управління якістю 52
3.4 Висновки 59
4 Програмна реалізація системи управління якістю розробки програмних продуктів 61
4.1 Вибір інструментарію та середовища реалізації 61
4.2 Програмна реалізація та тестування системи управління якістю розробки програмних продуктів 64
4.3 Висновки 74
Висновки 77
Перелік джерел посилання 79
Додаток А 83

Додаток Б..... 89
Додаток В 101

ВСТУП

У сучасному світі різнотипні інформаційні системи мають дуже широке розповсюдження – програмне забезпечення використовується всюди, від важкої промисловості до медичної сфери. Особливе місце у такому світі займає питання управління якістю програмного забезпечення.

Сьогодні поріг входу в ринок розробки програмного забезпечення є відносно низьким. Іншими словами, щоб стати постачальником програмних рішень, не потрібні великі інвестиції. З цієї причини на ринку немає дефіциту пропозицій з розробки програмних бізнес-систем. Проте, часто розроблюване програмне забезпечення має недостатньо високий рівень якості, що змушує замовників дуже уважно обирати виконавця.

Поняття якості програмного продукту є доволі комплексним, оскільки воно складається із множини різних характеристик, які бажають бачити кінцеві користувачі системи, проте всі вони включають в себе показники стабільності роботи, зручності використання, можливість повторного використання та легкої модифікації тощо. Таким чином, існує значний попит на неперервне удосконалення принципів та методів управління якості програмного забезпечення відповідно до появи нових технологій та стандартів.

Принципи та методи управління якістю при розробці програмних продуктів регламентовані за допомогою різних моделей розробки програмного забезпечення, які в останнє десятиліття стають все більше популярними серед невеликих груп розробників, а також найбільших представників ринку. Проте, сфера управління якості охоплює не лише галузь інженерії програмного забезпечення, а практично всі сучасні процеси проєктування, розробки та постачання будь-яких товарів та послуг – і кожній галузі притаманні свої особливості у застосуванні основних принципів, що регламентовані різними документами щодо систем менеджменту якості, найпоширенішим з яких є серія стандартів ISO 9000.

За останні роки методи управління якістю отримали широке використання та потужний розвиток у галузі інженерії програмного забезпечення, а також в інших

індустріях, в результаті значного попиту. Проте, оскільки поняття якості виражається у задоволенні вимог користувача, на даний момент відбуваються активні пошуки кращих стратегій ефективної розробки програмних продуктів та забезпечення їх якості.

Метою дослідження є удосконалення методу управління якістю розробки програмних продуктів.

Для досягнення мети потрібно виконати наступні завдання:

– провести аналіз моделей розробки програмного забезпечення із точки зору процесів управління якістю;

– проаналізувати системи менеджменту якості, вітчизняні та міжнародні стандарти;

– покращити метод управління якістю розробки програмних продуктів шляхом впровадження системи менеджменту якості;

– сформувати компоненти системи управління якістю розробки програмного забезпечення згідно серії стандартів ДСТУ ISO 9000 та принципів TQM;

– провести аналіз існуючих методів контролю якості програмного коду на предмет оптимізації;

– провести аналіз існуючих методів контролю якості організаційних процесів на предмет оптимізації;

– виконати проєктування програмної реалізації розробленої системи;

– розробити програмну реалізацію;

– провести тестування та аналіз запропонованого методу управління якістю розробки програмних продуктів.

Об’єктом дослідження є процеси життєвого циклу програмного забезпечення.

Предметом дослідження є методи та підходи управління якістю розробки програмних продуктів.

Наукова новизна отриманих результатів:

– отримав подальший розвиток підхід загального управління якістю у напрямку його застосування у сфері розробки програмних продуктів, що дозволяє підвищити якість програмного коду та процесів його розробки;

– удосконалено систему менеджменту якості TQM у сфері розробки програмних продуктів, що поєднує як засоби аналізу якості коду, так і методи статистичного аналізу якості процесів;

– удосконалено метод управління якості розробки програмних продуктів.

Практичне значення отриманих результатів полягає у вдосконаленні методу управління якістю розробки програмних продуктів, шляхом впровадження отриманої системи менеджменту якості у процес розробки, що дає змогу суттєво покращити ефективність роботи команди шляхом підвищення якості коду та оптимізації організаційних процесів.

Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи опираючись на результати емпіричних досліджень.

Теоретичними методами дослідження виступають: аналіз, синтез, абстрагування та порівняння.

Емпіричними методами дослідження виступають: опис та тестування.

За темою кваліфікаційної роботи опублікована наукова стаття «Порівняння метрик оцінки якості програмних продуктів» та опубліковані тези «Система управління якістю у розробці програмних продуктів» на конференції «Актуальні проблеми комп’ютерних наук АПКН-2021».

1 АНАЛІЗ СУЧАСНИХ МЕТОДІВ УПРАВЛІННЯ ЯКІСТЮ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

1.1 Аналіз сучасних моделей розробки програмного забезпечення

Якість є важливим атрибутом програмних продуктів оскільки її відсутність провокує втрати фінансів, часу, авторитету розробників та довіри користувачів. Дане поняття слід розглядати не лише із технічної точки зору, але і розуміти вплив якості програмного забезпечення на такі аспекти, як зручність використання та ступінь задоволеності кінцевого користувача. Більше того, широке використання інформаційних систем у різних галузях ринку зміщує фокус інженерії програмного забезпечення від безпосереднього створення рішення до задоволення всіх вимог зацікавлених осіб, що, у свою чергу, відображене у понятті якості програмного забезпечення. Тобто, якість програмного забезпечення – це одна із характеристик продукту, що відображає ступінь відповідності програмного забезпечення вимогам кінцевого користувача. Стратегія контролю якості, яка обрана невірно, може спричинити масу проблем при розробці продукту навіть при наявності кваліфікованих працівників.

Як результат значного попиту, за останні роки методи управління якістю отримали широке використання та потужний розвиток як у галузі програмної інженерії, так і в інших індустріях. На даний момент активно відбуваються пошуки кращих стратегій ефективної розробки програмних продуктів та забезпечення їх якості.

Сьогодні у багатьох компаніях розробка відбувається згідно однієї із моделей процесів розробки програмного забезпечення (software development process model). Процеси управління та контролю якості проводяться відповідно до методологій та конкретних моделей, за якими відбувається розробка. У роботах [1-3] до них відносять такі моделі:

а) класичні моделі розробки програмного забезпечення:

1. модель життєвого циклу програмного забезпечення;
2. прототипування програмного забезпечення;

3. спіральна модель;

б) об’єктно-орієнтована методологія;

в) моделі методології Agile.

Модель життєвого циклу програмного забезпечення (SDLC – software development life cycle) представлена у роботах [4-5] як лінійна послідовність усіх вагомих етапів побудови програмного продукту під час процесу розробки. Вона є класичною, проте має широке використання і у сучасних проєктах. Дана модель складається із наступних послідовних етапів:

– визначення вимог – формалізуються вимоги всіх зацікавлених осіб, середовище роботи програмного забезпечення та інтерфейси взаємодії із іншими інформаційними системами;

– аналіз – на цій фазі аналізуються вимоги до продукту та проводиться діяльність із забезпечення якості у вигляді огляду та оцінки проєкту експертами;

– проєктування – описуються детальні визначення вхідних і вихідних даних, а також обчислювальних процедур всередині програми;

– реалізація – втілення результатів проєктування за допомогою програмного коду. На даному етапі управління якістю виражене лише у перевірці програмного коду експертами;

– тестування – етап, що призначений для діяльності із управління та забезпечення якості продукту, проводяться різноманітні тести для вимірювання вагомих атрибутів системи, таких як стабільність, швидкість роботи, складність коду, тощо. Важливою частиною процесу є модульне тестування (англ. unit testing), що полягає в тестуванні кожного модуля коду окремо. Також для перевірки якості програмного забезпечення використовують спеціальні метрики. Даний етап життєвого циклу проводиться після завершення фази реалізації;

– введення у дію – програмне забезпечення вводиться у експлуатацію на апаратному забезпеченні замовника;

– супровід – фінальний етап життєвого циклу, який полягає у адаптації продукту та корекції роботи у разі необхідності.

Із точки зору управління якістю, дана модель має вагомі недоліки, оскільки контроль якості коду відбувається на пізніх етапах розробки, а до етапу тестування є представленим лише за допомогою ручного перегляду коду фахівцем. Хоча для тестування виділений окремий етап, проте він проводиться лише по завершенню написання всього коду.

У роботі [6] описана модель прототипування програмного забезпечення (prototyping model). Дана модель є ітеративною, в якій на кожній ітерації розробки програмного забезпечення створюється його прототип. На першій ітерації лише частина вимог втілюється у продукті, а кожна наступна ітерація використовується для розробки нового прототипу, який реалізовує наступну частину вимог – даний процес продовжується до того моменту, як всі вимоги будуть задоволеними. Кожен із прототипів тестується та оцінюється групою розробників, експертів та кінцевих користувачів. У дослідженні [7] модель прототипування визнається більш ефективною ніж модель SDLC, оскільки проєкти, розроблені за даною методологією використовували менше ресурсів при розробці та продукували менше надлишкового коду, проте програми, розроблені за моделлю життєвого циклу програмного забезпечення мали більш розгорнуту функціональність.

Відповідно роботі [1], основні переваги та відмінності моделі прототипування від SDLC полягають у інтенсивному залученні користувачів, клієнтів та експертів по якості програмного забезпечення у процес розробки. За допомогою цією особливості досягається нижча ймовірність системних збоїв та краще розуміння роботи системи в цілому, оскільки всі проблеми, невірні рішення та неточності усуваються швидко та на ранніх етапах циклу розробки продукту. Даний підхід є більш оптимальним з точки зору методів управління якістю, проте він обмежує свободу розробника на впровадження змін та нововведень в інформаційну систему.

Спіральна модель (spiral model) представляє собою ітеративну методологію яка забезпечує ефективність виконання кожної фази моделі SDLC. Згідно із роботами [8-9], спіральна модель розробки програмного забезпечення включає інтеграцію побажань клієнта і зміну вимог, аналіз ризиків та рішень, а також

інженерію якості на кожній фазі моделі життєвого циклу програмного забезпечення. На кожному глобальному етапі розробки може використовуватись одна чи більше ітерацій спіральної моделі для забезпечення якості продуктів.

Дана модель вважається більш оптимальною для великих та середніх проєктів, оскільки вона приділяє більше уваги комунікації із кінцевими користувачами та підтриманні зворотного зв'язку.

Перевагами даної методології згідно досліджень [8-9] є більша ймовірність отримання системи, що найкраще задовольняє вимоги замовника, не виходячи при цьому за рамки фінансового бюджету та часових ресурсів. Дані характеристики і є головними критеріями успішного управління якістю розроблюваного проєкту, що можна віднести до головних переваг даної моделі. До того ж, даному методу властивий високий рівень документації продукту, гнучкість у зміні вимог користувача та акцент на управлінні ризиками.

Із недоліків варто виділити неможливість застосування методології у малих проєктах, оскільки вона залучає додаткові організаційні процеси і, відповідно, додаткове використання всіх типів ресурсів. Вона є складнішою за інші класичні моделі розробки, оскільки не передбачає кількість ітерацій спіралі, а отже іноді важко визначити кінець розробки як модулів програми, так і усього продукту в цілому – із цього випливають складності у менеджменті часу, оскільки будь-які оцінки тривалості роботи неможливо дати із прийнятною точністю.

Об'єктно-орієнтована модель розробки (object-oriented model) суттєво відрізняється від попередніх класичних моделей, оскільки вона опирається на ідею про інформаційну систему, як колекцію незалежних одиниць, що називаються класами [1]. Кожен клас присвячений окремій фізичній чи інакшій сутності і зберігає дані на програмні функції, необхідні для реалізації даної сутності. По суті, клас представляє собою набір атрибутів – характеристик сутності та набір методів, що є функціями класу, програма представляє собою систему взаємодії об'єктів, що є конкретними реалізаціями того чи інакшого класу.

Відповідно, відрізняються і методи управління та забезпечення якості. Особливості застосування об'єктно-орієнтованої методології із даної точки зору

розглянуті у роботах [1, 10]. Оскільки в основі об’єктно орієнтованого програмування лежать поняття абстракції, інкапсуляції, успадкування та поліморфізму, то і безпосередньо якість написаного коду залежить від того, наскільки добре дані концепції реалізовані в архітектурі програмного коду. Адекватність реалізації даних принципів у програмі є однією із найважливіших компонентів управління якістю об’єктно-орієнтованих інформаційних систем, що забезпечує легкість читання коду, простоту підтримки та стабільність роботи. Використання класів забезпечує високий ступінь повторного використання програмного коду та просту інтеграцію розроблених чи існуючих бібліотек у проєкт, що підвищує якість кінцевого продукту [1].

Процес розробки, так само, як і процес управління якістю, адаптований до структури, скомпонованої із класів, а тому суттєво відрізняється від розробки за інакшими методологіями. Згідно із роботою [11], даний процес складається із наступних етапів:

– аналіз. На даному етапі будуються класи, які необхідні для виконання вимог щодо атрибутів та функціональних елементів програми. До діяльності стосовно управління якістю під час аналізу відносять огляд проєктування (DR – design review), що включає в себе об’єктно-орієнтований аналіз (OOA – object-oriented analysis);

– проєктування. Загалом, даний етап необхідний для деталізації та покращення структури класів, розробленої на фазі аналізу, для подальшої фіксації написаного коду у програмних бібліотеках для повторного використання у майбутньому. В огляд на цій стадії розробки входить тестування інформаційної системи згідно методу об’єктно-орієнтованого проєктування (англ. OOD – object-oriented design), що передбачає оцінку рівня детальності класів, відповідності вимогам користувача та визначення адекватності дотримання відповідності основним принципам об’єктно-орієнтованого програмування;

– реалізація. Після опису класів інформаційної системи, відбувається реалізація їх структури у програмному коді, а також реалізація необхідних методів класів та інтерфейсів взаємодії;

– тестування. На стадії тестування проводиться тестування класів (class testing), інтеграційне тестування (integration testing) та системне тестування (system testing). Тестування класів еквівалентне модульному тестуванню у класичних моделях розробки. Інтеграційне тестування полягає у перевірці коректності роботи різних кластерів класів із послідовним їх об’єднанням. Системне тестування проводиться для фіксації проблем функціонування системи в цілому;

– введення у дію та супровід. Дані етапи повністю відповідають таким у моделі життєвого циклу програмного забезпечення.

Очевидно, об’єктно-орієнтована модель розробки має ряд переваг відносно інших моделей. За рахунок використання класів, код можна багаторазово застосовувати, що призводить до зменшення ціни проєкту та часу, необхідного для розробки. Простіше розробляти та підтримувати великі проєкти, проте розробка за даним методом складніша, оскільки потребує додаткових навичок програмування, а додатки, побудовані на основі об’єктно-орієнтованої архітектури зазвичай працюють повільніше за аналоги, побудовані на функційних чи процедурних мовах програмування. Також, даний підхід не завжди підходить для вирішення задачі.

Із точки зору процесів управління якістю, дана модель є доволі оптимальною, передбачаючи активне тестування на багатьох фазах розробки та застосовуючи методи контролю якості ще на етапі проєктування класів та їх структури.

Методологія Agile – набір концептуальних поглядів та практик, які покликані підвищити ефективність усіх процесів розробки програмного забезпечення. З точки зору програмної інженерії, методи Agile передбачають ітеративну розробку, де вимоги та рішення розвиваються через взаємодію самоорганізованих різнофункціональних команд розробників.

У межах методології Agile було створено багато популярних моделей розробки програмного забезпечення, які розділяють єдині принципи та підхід щодо процесів управління якістю на різних етапах життєвого циклу. До таких моделей відносяться наступні [14]: метод динамічного розвитку системи (DSDM – dynamic system development method), модель RAD (RAD – rapid application development),

Екстремальне програмування (XP – extreme programming), функціонально-орієнтована розробка (FDD – function driven development) тощо.

Типовий процес розробки програмного забезпечення згідно із методологією Agile включає наступні діяльності:

– вимоги до проєкту на початку розробки не є чітко визначеними і базуються на прикладах використання, наведених замовником – на це також опираються орієнтовний бюджет та час, відведені на розробку продукту;

– маленькі групи розробників починають реалізацію із введення найбільш пріоритетних функцій та інтегрують нові функції у вигляді нових елементів прогами – інкрементів;

– аналіз та проєктування включені в етап реалізації;

– групи розробників проводять проєктування, аналіз, реалізацію та тестування не передаючи продукт іншій групі;

– заохочуються щоденні зустрічі розробників та участь клієнтів у процесі розробки.

До обмежень моделей методології Agile можна віднести [14] потребу у високому рівні кваліфікації розробників для виконання роботи у групі, контроль більшої частини процесів циклу розробки однією командою може спричинити складності в ідентифікації проблем. Також, недостатня увага приділяється документації, що може викликати проблеми із підтримкою, супроводом та подальшою інтеграцією нових модулів у програму.

З точки зору управління якістю, методологія Agile є ефективною, оскільки моделі розробки є ітеративними, а кожна ітерація передбачає ряд процесів контролю якості. Це досягається шляхом поділу розробників на маленькі групи і до них відносяться перегляд та оцінка результатів аналізу та проєктування, перевірка коду та планування модульного тестування ще до реалізації програмного коду.

Порівняно із об’єктно-орієнтованою моделлю та класичними моделями розробки програм, процеси забезпечення якості у Agile мають певні переваги та недоліки:

– процеси забезпечення якості програмного коду проводяться на основі інкрементів, які включають мале число нових функцій та представляють невелику частину від загальних вимог до продукту, що інколи заважає провести ефективне тестування продукту в цілому;

– у моделях Agile представлене активне залучення замовників до співпраці, що гарантує кращу відповідність вимогам, на відміну від конкуруючих моделей;

– інкременти розробляються невеликою командою розробників, що виключає допомогу незалежних фахівців та тестувальників.

Таким чином, не існує найкращої моделі розробки програмного забезпечення, оскільки використання того чи інакшого підходу передбачає свої переваги та недоліки. У реальних продуктах найбільш доцільним є використання принципів різних моделей та методологій відповідно до цілей розробки програмного продукту [15]. Однак, можна прослідкувати певні тенденції відносно організації процесів управління якістю програмного забезпечення та ступеня задоволення потреб зацікавлених сторін. Впровадження аналізу та оцінки ефективності кожного етапу життєвого циклу, заохочення розробників до прямої співпраці між собою та із замовником, а також використання різних типів заздалегідь запланованих тестувань підвищують ймовірність розробки продукту високої якості.

1.2 Аналіз систем менеджменту якості

Дослідження літературних джерел, що описують моделі розробки програмного забезпечення включно з підходами до виконання процесів управління якістю, проведене у попередньому розділі, показує, наскільки різноманітною та актуальною є дана предметна область. Проте, для кращого розуміння ситуації, слід дослідити те, яким чином відбувається управління якістю в інших індустріях, таких як, наукові розробки, різні види виробництва та промисловості тощо.

Система менеджменту якості (QMS – quality management system) – це система, що документує політики, бізнес процеси та процедури, необхідні

організації для того, щоб розробляти та постачати її товари клієнтам і підвищувати рівень задоволення клієнтів за допомогою високої якості продукції [16]. QMS включають в себе як методології із концепціями, так і певні конкретні засоби покращення якості у різних галузях.

У роботі [17] описано найбільш відомі та актуальні системи менеджменту якості:

- загальне управління якістю;
- розгортання функції якості;
- тестування продуктивності;
- шість сигм;
- ощадливе виробництво.

Загальне управління якістю (TQM – total quality management) – підхід у сфері менеджменту якості, що полягає у неперервному підвищенні якості всіх організаційних процесів [19]. Основні положення TQM описано у роботі [18], які включають постійне паралельне покращення трьох складових:

- якості продукції;
- якості організаційних процесів;
- рівня кваліфікації персоналу.

Поняття якості як такої визначається за допомогою наступних категорій:

- ступінь реалізації вимог клієнтів;
- значення фінансових показників компанії;
- рівень задоволеності працівників компанії своєю роботою.

Головна ідея загального управління якістю полягає в тому, що компанія повинна працювати не лише над якістю продукції, але і над якістю організації роботи в компанії, включаючи роботу персоналу [18-19].

При використанні TQM, компанії беруть за основу універсальні принципи Демінга, Трилогію Джурана або Чотири правила менеджменту якості, що представляють собою конкретні втілення даної методології. Схожі положення зафіксовані і у офіційних документах управління продовольства та медикаментів США (Food and Drug Administration – FDA) [16]. Отже, можна зробити висновок,

що всі системи менеджменту якості, що засновані на принципах TQM, включають три основні компоненти: філософію менеджменту, модель чи процес покращення та набір засобів контролю якості.

Філософія менеджменту (management philosophy) – загальні принципи підтримки якості, забезпечення оптимальних умов роботи для працівників та способи роботи із замовником. Також, до філософії входять структура побудови виробничого процесу, способи перевірки рівня кваліфікації працівників та ієрархія посад у компанії.

Модель покращення (improvement model) представляє собою збірне поняття, яке включає всі способи перевірки, тестування та покращення різних аспектів виробничої діяльності від оцінки та удосконалення різних характеристик розроблюваного продукту до вказівок щодо виконання плану покращення.

Засоби контролю якості (quality Control (QC) tools) – формалізований набір засобів, як правило, заснованих на статистичних методах, що дозволяють оцінити рівень якості організаційних процесів та рівень якості самого продукту.

У роботах [20-21] описано емпіричне статистичне дослідження, де на основі результатів роботи реальних компаній була доведена ефективність використання загального управління якістю у різних видах діяльності, показуючи значний ріст продуктивності та загальної якості результатів роботи компаній. В цілому, даний підхід успішно показав себе у реальному використанні і отримав багато нових реалізацій – фактично, усі сучасні системи менеджменту якості беруть за основу принципи, описані у TQM.

Перед безпосередньо аналізом наступних актуальних систем менеджменту варто розглянути основні положення стандартів серії ISO 9000. ISO 9000 – серія міжнародних стандартів, що містять терміни та визначення, загальні принципи менеджменту якості, вимоги до систем менеджменту якості організацій та підприємств, а також рекомендацій для досягнення стійкого результату [22]. Дані стандарти неодноразово переглядались, так перша версія була підготовлена у 1987 році, проте, остання на даній момент, п’ята версія ISO 9000 була затверджена у 2015 році для відповідності сучасним умовам ринку – в Україні дана серія офіційно

затверджена як серія ДСТУ ISO 9000 у 2016 році [24]. В основі даної серії стандартів лежать ідеї та положення методології загального управління якістю та відгалуження стандартів забезпечення якості у сфері військово-промислового комплексу [17]. Серія міжнародних стандартів та українське видання представлене у літературних джерелах [23] та [24] відповідно.

Серія ДСТУ ISO 9000 складається із трьох основних документів:

– ДСТУ ISO 9000 – «Основні положення та словник термінів»;
– ДСТУ ISO 9001 – «Вимоги»;
– ДСТУ ISO 9004 – «Настанови щодо досягнення сталого успіху».

Слід розуміти, що серія ISO 9000 – це не стандарти якості, а стандарти систем управління якістю, тобто перелік вимог та рекомендацій для виробництв та постачальників послуг, задовольнивши які, організації забезпечують високу ефективність саме своїм системам менеджменту якості. Загалом, згідно емпіричного дослідження [25], дана серія стандартів приносить ряд корисних результатів компаніям із будь-якої галузі як у короткотривалій, так і у довготривалій перспективі.

Розгортання функції якості (QFD – quality function deployment) описане у роботі [17]. Дана система представляє собою структурований процес планування проєктування нового продукту чи послуги із акцентом на розуміння того, що замовник хоче або потребує. Потім ці бажання трансформуються в характеристики продукту чи послуги та в деталі процесів всередині самої компанії. До переваг QFD можна віднести менший об’єм часу, необхідного для проведення проєктування та, відповідно, нижчу ціну всього проєкту в цілому. Дана система менеджменту якості потребує поділ компанії на команди, кожна з яких не має своєї спеціалізації і може виконувати різноманітні функції. Головною особливістю є використання специфічного методу під назвою Будинок якості (the house of quality), що є особливим методом роботи із вимогами користувача та їх перетворення у характеристики якості фінального продукту. Проте, через дану особливість, метод розгортання функції якості не є універсальним і підходить не для всіх сфер виробництва товарів та послуг.

Поширена концепція підходу до управління якістю під назвою тестування продуктивності (benchmarking) полягає у порівнянні робочих практик, процесів та стратегій із аналогічними практиками, процесами та стратегіями, що використовуються в інших компаніях з метою інтеграції найоптимальніших ідей у власний виробничий процес. Порівняння може бути застосоване як до готових товарів і послуг, так і до усіх процесів, що протікають в організації. Також, ефективною практикою є порівняння різних елементів на предмет різниці в плані ефективності всередині самої компанії.

Очевидними перевагами тестування продуктивності відносно інших технік покращення якості є те, що прогрес відбувається не маленькими поступовими кроками, а великими стрибками – одразу адаптуючи чужі ефективні методи, не розробляючи їх з нуля. Звідси і випливає основний ризик – аналітики компанії можуть зробити не вірне рішення і перейняти неефективні методи та техніки в свою організацію.

Інакшою реалізацією принципів загального управління якістю є методологія Шість сигм (Six Sigma), що описана у роботах [17,26]. На відміну від інших систем менеджменту якості, що беруть за основу TQM, основні положення у методології Шість сигм концентруються на використанні аналітиками та менеджерами компанії статистичних даних та результатів статистичного аналізу для вдосконалення виробництва і зменшення кількості дефектів. Найважливішою ціллю стоїть зниження частоти відхилень характеристик якості продуктів однакового типу від еталонних показників на виробництві.

До показників продуктивності додаються нові: сигма рівні – ступінь середнього відхилення по всіх атрибутах продукту, кількість дефектів на мільйон – відображає частоту появи даних відхилень, ціна поганої якості – об’єм втраченого часу, фінансів та покупців.

Останньою із систем, розглянутих у даній роботі, є ощадливе виробництво (lean manufacturing). Відповідно до дослідження [27], дана методологія фокусується на мінімізації витрат всередині систем виробництва, одночасно максимізуючи ефективність. Діяльності оптимізуються за допомогою методів ощадливого

виробництва або класифікуються як такі, що не створюють додаткової цінності і видаляються із виробничого процесу. Головним елементом оцінки важливості процесу є цінність для користувача, процеси класифікуються згідно їх впливу на якість кінцевого результату.

До негативних елементів виробництва відносяться дефективні продукти, надлишкове виробництво, процеси, що не створюють цінності, транспортування товарів, надмірний рух робітників, інструментів, товарів, а також очікування. На предмет використання виробничих ресурсів аналізуються процеси, практики та стратегії компанії, результати роботи – згідно результатів аналізу проводяться покращення за допомогою методів, що входять конкретно до ощадливого виробництва та загальних статистичних методів, притаманних усім системам менеджменту якості [17]. За результатами дослідження [28], компанії, що використовували принципи ощадливого виробництва, отримали підвищення ефективності роботи і, відповідно, прибутків.

Результати проведеного аналізу систем менеджменту якості приводять до схожих висновків, як аналіз моделей розробки програмного забезпечення. Не існує найоптимальнішої системи – стратегія управління якістю повинна бути підібрана з розумінням аспектів процесів виробництва чи розробки програмного забезпечення, повинні бути враховані цілі та їх пріоритети. Головні принципи усіх систем управління якістю базуються на положеннях TQM та ISO 9000, найголовнішим із яких є те, що для адекватного покращення якості розроблюваного, компанія повинна працювати не лише над якістю продукції, але і над якістю організації усіх процесів виробництва та розробки. Згідно із дослідженнями, наведеними вище, стратегія TQM забезпечує хороші результати, проте дані принципи не регламентовані методологіями розробки програмного забезпечення, що є їх недоліком. Також, слід звернути увагу на те, що згідно моделей розробки програмного забезпечення, процеси управління якістю лежать на плечах розробників і тестувальників, а у розглянутих системах менеджменту якості у даних процедурах беруть участь ще й аналітики та менеджери.

1.3 Аналіз засобів контролю якості

У попередніх підрозділах було здійснено аналіз підходів до проведення управління якості у моделях розробки програмного забезпечення та в інших індустріях. Інакшою важливою складовою менеджменту якості є вибір правильних методів вимірювання та оцінки процесів, а також вибір оптимальних інструментів їх покращення. Відповідно до предметної області, проведемо аналіз засобів контролю якості безпосередньо програмного коду та програмного продукту в цілому, а після цього проведемо аналіз наявних засобів контролю якості організаційних процесів. Засоби контролю якості програмного забезпечення описані у роботах [29-31], засоби контролю якості організаційних процесів наведені у роботах [16-18].

Вірогідно, найбільш поширеним із даних методів є огляд коду (code review). Це – процес забезпечення якості програмного забезпечення під час якої один розробник або група розробників перевіряють код програми в основному за допомогою перегляду та читання. Обов’язковою умовою є те, що хоча б один із учасників повинен не бути автором даного коду.

Огляд коду виконують із метою покращення якості коду, знаходження та виправлення дефектів, перевірки на відповідність міжнародним стандартам та стандартам всередині компанії. Також важливими цілями є обмін досвідом та знанням, пошук кращих рішень та створення відчуття спільної відповідальності за результати роботи всієї команди.

Тестування програмного забезпечення проводиться для аналізу відповідності продукту вимогам, що висуваються. Дана практика включає в себе запуск всієї системи або програмного модуля для оцінки однієї чи більше характеристики. Загалом, від програмного коду вимагають коректну роботу, простоту, використання ненадмірного числа змінних та функцій, відсутність помилок та використання оптимальних алгоритмів, також якісний код має бути таким, щоб його можна було легко прочитати та зрозуміти. Тестування допомагає виявити проблеми на ранніх етапах розробки і запобігти небажаним наслідкам їх наявності.

Існує багато систем класифікацій тестування програмного забезпечення, найбільш поширеною з них є класифікація за рівнями тестування. За нею тестування поділяють на три види: тестування окремих компонентів, при якому тестуються класи та функції, інтеграційне тестування, при якому тестуються інтерфейси між компонентами, системами та підсистемами, та системне тестування, при якому інтегрована система тестується на предмет рівня задоволення всіх вимог кінцевих користувачів.

Ще одним засобом є метрики програмного забезпечення – це стандартизовані вимірювання певних характеристик програмної системи чи процесу із метою визначення їх ступеня та міри впливу на продукт у цілому. Метою обрахунку програмних метрик є отримання об’єктивних та зрозумілих вимірювань для подальшого використання при плануванні бюджету, плану розробки, тестуванні, оптимізації та документації програмного забезпечення.

Використання даних метрик надає ряд переваг для розробників, оскільки це спрощує прийняття вагомих рішень щодо архітектури програмного продукту, використання тих чи інакших мов програмування та фреймворків. Також, це дозволяє швидко розпізнавати та усувати можливі несправності продукту, а також дає гарне розуміння швидкодії програми та шляхи її оптимізації. Тим не менш, оскільки процес проєктування та розробки програмного продукту є доволі складним і комплексним, метрики програмного забезпечення виступають лише додатковим інструментом при аналізі проєкту спеціалістом, та мають певні обмеження [31].

До поширених типів метрик програмного забезпечення можна віднести кількісні метрики, метрики розміру, метрики потоку керування програмою, метрики потоку керування даними, об’єктно-орієнтовані метрики та метрики динамічної зв’язності. Їх класифікація наведена на рисунку 1.1.

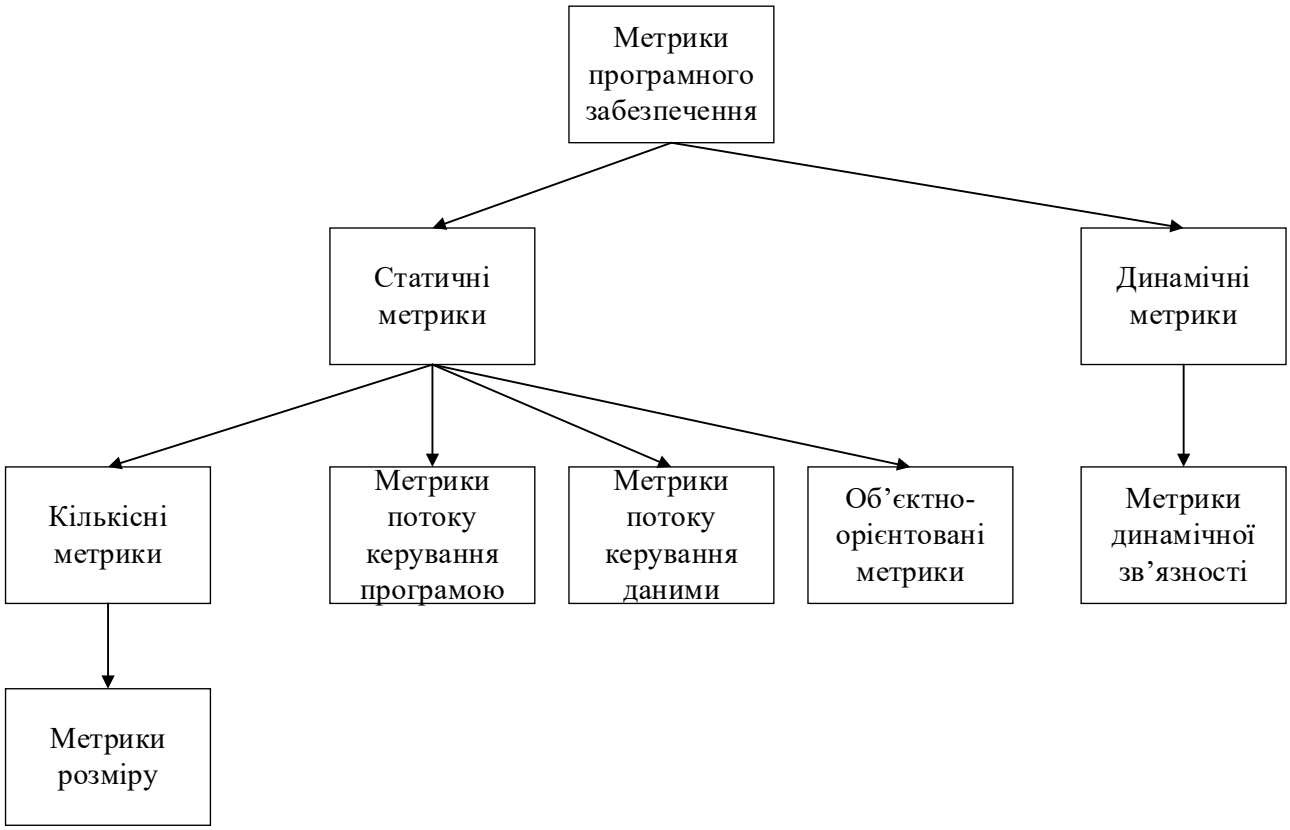


Рисунок 1.1 – Класифікація метрик програмного забезпечення

Слід виділити окремо діяльність у напрямку розвитку м'яких навичок (soft skills) співробітників. Під м'якими, або гнучкими, навичками розуміють набір умінь та навичок, які не пов'язані із безпосередньо роботою робітника. До них відносять знання трудової етики, вміння працювати в команді, вирішувати нестандартні задачі, здатність до ефективної комунікації з метою поширення та засвоєння нового досвіду, володіння іноземними мовами тощо. Дані знання та вміння не впливають безпосередньо на якість коду, проте вони є доволі важливими для підтримки ефективної командної роботи та росту професіональної компетентності усіх працівників. Багато компаній у сфері розробки програмних продуктів виділяють час та кошти для розвитку гнучких навичок у працівників і це приносить корисні результати.

У системах менеджменту якості використовується ряд засобів контролю якості, що застосовуються до організаційних процесів[17]. До них відносяться такі методи контролю якості, які підходять для людей без специфічних знань в галузі управління якістю та статистиці, а також тому що вони допомагають вирішувати

більшість проблем, пов’язаних із якістю організаційних процесів. Одними із представників даних засобів є: причинно-наслідкова діаграма (cause-and-effect diagram), відома також як діаграма Ішикави (Ishkawa diagram), контрольна таблиця (check sheet), контрольна карта (control chart), відома також як діаграма Шухарта (Shewhart chart), гістограми, діаграма Парето (Pareto chart) та інші.

Зрозуміло, що частину із наведених вище засобів контролю можна без проблем автоматизувати, наприклад, розрахунок програмних метрик та розрахунок статистичних методів контролю якості. Даний набір функцій доцільно інтегрувати у сучасний засіб управління проєктами, що зробить процес управління якістю швидшим та зручнішим.

Дані засоби мають різне призначення і мету, в реальних проєктах доцільно комбінувати їх для того, щоб керувати якістю усіх процесів, що протікають в компанії. Даний підхід забезпечить як високу якість кінцевого продукту, так і низькі витрати на розробку, як наслідок ефективного керування проєктом в цілому.

1.4 Постановка задачі

Забезпечення високої якості програмного продукту є одним із основних критеріїв успішного проєкту – і чим більший проєкт, тим більшу вагу має кожен аспект продукту. Для досягнення оптимальних результатів, компанії використовують різні стратегії із управління якістю програмного забезпечення. Сьогодні, усі популярні методології розробки програмного забезпечення формалізують принципи та конкретні засоби контролю якості, що формує основу для використання методів управління якістю при розробці.

У результаті порівняльного аналізу сучасних моделей розробки програмного забезпечення з точки зору сфери управління якістю та систем менеджменту якості, інтегрованих в інших індустріях, було виділено недоліки актуальних методологій управління якістю в сфері розробки програмних продуктів.

Емпіричні дослідження [20, 21, 28, 33], розглянуті в ході аналізу предметної області, приводять до висновків про високу ефективність застосування принципів

TQM та ДСТУ ISO 9000 у всіх сферах промисловості, тому дані принципи варто інтегрувати і в процеси управління якістю, що практикуються сьогодні при розробці програмного забезпечення. Актуальні на даний момент підходи не звертають увагу на покращення організаційних процесів всередині компаній, що веде до неефективного використання ресурсів та зниження якості кінцевого програмного продукту, проте застосування описаних вище принципів, за допомогою розробки системи менеджменту якості відповідно до серії стандартів ДСТУ ISO 9000 може вирішити дану проблему. Аналіз засобів контролю якості коду та процесів розробки привів до висновків про можливу автоматизацію частини засобів із подальшою інтеграцією в розроблювану систему управління якістю розробки програмного забезпечення.

Об’єктом дослідження є процеси життєвого циклу програмного забезпечення.

Предметом дослідження є методи та підходи управління якістю розробки програмних продуктів.

Метою дослідження є удосконалення методу управління якістю розробки програмних продуктів.

Для досягнення мети потрібно виконати наступні завдання:

– покращити метод управління якістю розробки програмних продуктів шляхом впровадження системи менеджменту якості;

– сформувати компоненти системи управління якістю розробки програмного забезпечення згідно серії стандартів ДСТУ ISO 9000 та принципів TQM;

– провести аналіз існуючих методів контролю якості програмного коду на предмет оптимізації;

– провести аналіз існуючих методів контролю якості організаційних процесів на предмет оптимізації;

– виконати проєктування програмної реалізації розробленої системи;

– розробити програмну реалізацію;

– провести тестування та аналіз результатів.

Розроблений підхід та підібрані методи дозволять позбутись від проблем існуючих методологій розробки та досягти більшої ефективності роботи та вищої якості програмних продуктів.

1.5 Висновки

У першому розділі кваліфікаційної роботи було виконано теоретичний аналіз сучасного стану сфери управління якістю у розробці програмних продуктів та за її межами. Були виявлені переваги на недоліки існуючих рішень відносно один одного та сформовано задачу на розробку шляху удосконалення.

У підрозділі 1.1 був проведений порівняльний аналіз моделей розробки програмного забезпечення та визначено порядок проведення процесів контролю якості в них. У результаті порівняння були описані основні особливості, переваги та недоліки кожної з моделей.

У підрозділі 1.2 був здійснений порівняльний аналіз систем менеджменту якості, що використовуються в інших індустріях, здебільшого у сфері масового виробництва та проєктування продуктів для масового споживача. Також були описані міжнародні і вітчизняні серії стандартів систем менеджменту якості та було здійснене порівняння із підходами, що використовуються при розробці програмних продуктів з метою знаходження шляхів удосконалення.

У підрозділі 1.3 був здійснений опис та аналіз методів контролю якості, порівняння сфери їх застосування та здійснено аналіз на предмет їх покращення та автоматизації.

У результаті, були обґрунтовані принципи вдосконалення методу управління якості у розробці програмних продуктів та сформовані вимоги до розроблюваної інформаційної системи.

2 МОДЕЛЬ ТА МЕТОДИ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

2.1 Компоненти системи управління якістю розробки програмних продуктів

Із попередньо проведеного аналізу різних аспектів управління якості при розробці програмного забезпечення та у інших індустріях були зроблені висновки щодо недосконалості методів, регламентованих моделями розробки програмного забезпечення. Кожна із них, включаючи об’єктно-орієнтовану методологію, класичні моделі розробки та інші, мають ряд недоліків відносно способів організації управління якістю у галузі масового виробництва товарів та послуг. Для вирішення даної проблеми слід розробити систему управління якістю розробки, що бере за основу систему TQM, для програмних продуктів, що включатиме в себе як основні принципи управління якістю та організації роботи, так і прикладні засоби контролю якості із різних точок зору. Згідно дослідження [33], систему управління якістю слід розроблювати саме відповідно до положень стандартів серії ДСТУ ISO 9000, оскільки так можна досягти найкращих результатів. Результати використання таких систем зображено нижче на рисунку 2.1.

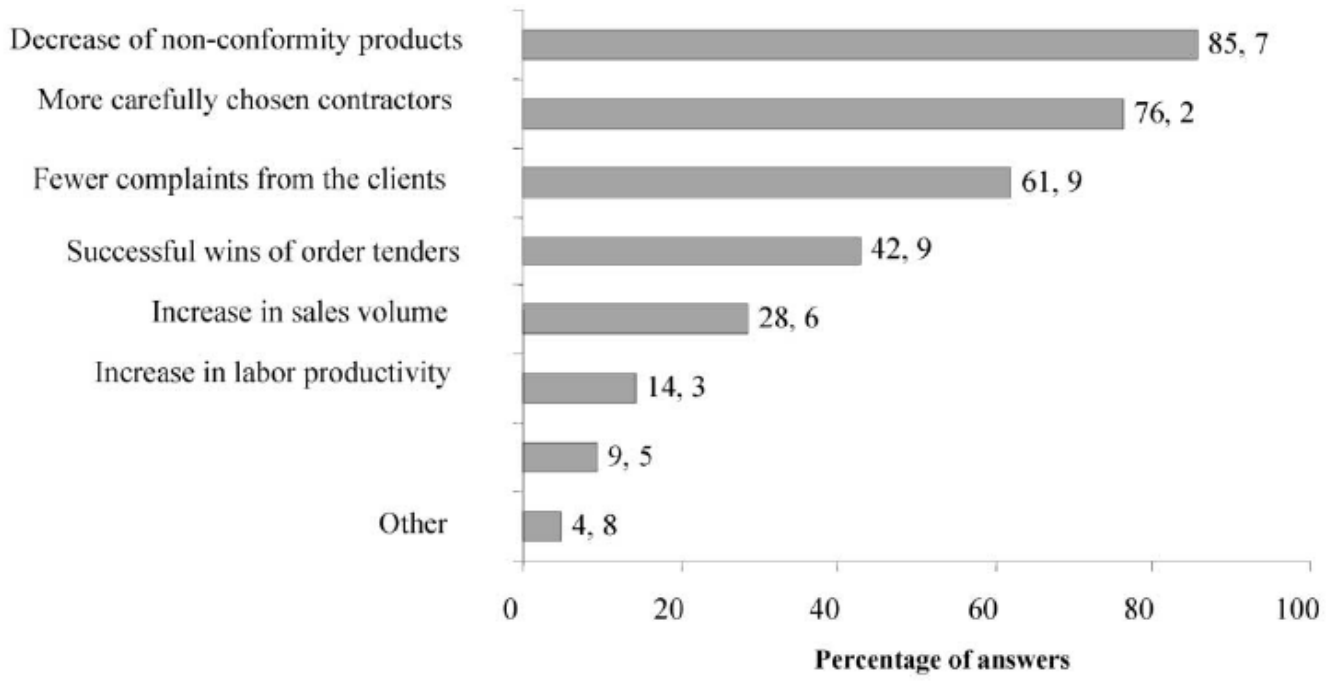


Рисунок 2.1 – Результати впровадження систем управління якістю

Дана система управління якістю відповідає положенням серії стандартів ДСТУ ISO 9000, які зображені на рисунку 2.1, відповідно, вона включає наступні принципи [23]:

а) орієнтованість на вимоги користувача:

1. розуміння потреб існуючих та майбутніх користувачів;
2. вимірювання задоволеності клієнта;
3. бажання перевищити очікування користувача;

б) керівництво:

1. визначення адекватних цілей;
2. встановлення довіри;
3. визначення напряму роботи компанії;

в) залучення людей:

1. заохочення обміну досвідом;
2. впровадження засобів ефективної співпраці;
3. оцінка продуктивності кожного члена команди;

г) процесний підхід:

1. пріоритет на процесах покращення;
2. неперервний пошук та оптимізація надлишкових процесів;
3. визначення зв’язків між процесами;

д) покращення:

1. заохочення всіх працівників до покращень;
2. неперервна оцінка та аналіз ефективності;

е) прийняття рішень на основі доказів:

1. забезпечення доступу до точних та надійних даних;
2. використання правильних методів аналізу даних;
3. підтримання балансу між результатами аналізу та практичним досвідом;

ж) управління відношеннями:

1. поділ відношень на короткострокові та довгострокові;
2. неперервна оцінка якості роботи із партнерами.

Рисунок 2.2 – Принципи управління якістю згідно ДСТУ ISO 9000

Філософія загального управління якістю передбачає неперервний аналіз якості всіх аспектів діяльності, що добре підходить до організаційної структури розробки програмних продуктів. Це не дивно, оскільки підвищення якості роботи програміста шляхом оптимізації його робочого плану та умов його роботи має безпосередній вплив на якість результатів його роботи. Аналіз організаційної структури підприємства пов’язаний із ефективним керуванням ресурсами компанії, тобто при оптимальному використанні робочого часу працівника з’являється можливість виділення ресурсів на підвищення його кваліфікації та покращення навичок роботи в команді.

Для ефективної роботи команди, діяльність щодо покращення навичок працівників повинна бути системною та запланованою: тренінги щодо покращення м’яких навичок, регулярні зустрічі членів команди, опитування працівників на

предмет пропозицій тощо, а проведення даних видів діяльності вимагає правильний розподіл ресурсів. Отже, керування ресурсами та неперервний аналіз роботи компанії із подальшим покращенням результатів є одними із ключових частин розроблюваної системи.

Головним критерієм правильної системи управління якістю є регламентація як методів контролю якості продукту, так і методів контролю якості процесів. Повинні бути конкретизовані методи оцінки якості коду та методи оцінки якості процесу написання коду – дані методології складають наступні два компоненти системи управління якістю.

Отже, можна виділити п’ять основних компонентів розроблюваної системи управління якістю розробки програмних продуктів: дотримання принципів управління якістю, неперервний аналіз, керування ресурсами, підвищення якості продукту та підвищення якості процесів. Дані компоненти зображені на рисунку 2.3. Така система є універсальною і не залежить від підібраної моделі розробки програмного продукту, але охоплює управління якістю усіх процесів, що проходять у компанії.

Рисунок 2.3 – Модель системи управління якістю, що втілює принципи TQM

Стандарти серії ДСТУ ISO 9000 вимагають також наявність інструментів забезпечення дотримання даних компонент в процесі діяльності компанії [24]. Тому метою кваліфікаційної роботи, окрім удосконалення системи управління якістю TQM, є також розробка інформаційної системи, що включатиме в себе весь необхідний інструментарій. Для ефективного керування ресурсами та аналізу ефективності слід розробити програмний комплекс управління проєктом. Для підвищення якості продукту та процесів необхідно використовувати відповідні засоби, що будуть описані у наступних розділах.

2.2 Засоби контролю якості програмного забезпечення

Програмний код є представленням певного алгоритму за допомогою інструментів мови програмування, а тому його характеристики є доволі комплексними і складними для перевірки. Метрики програмного забезпечення представляють собою чисельні значення характеристик програмного коду та використовуються для аналізу різних аспектів продукту з метою передбачення можливих проблем. Програмні метрики можуть бути використані на будь-якому етапі життєвого циклу програмного забезпечення. Наприклад, на етапі аналізу вимог та постановки задачі, метрики допомагають оцінити необхідний обсяг робіт та, відповідно, фінансові і трудові ресурси. Також метрики можуть використовуватись безпосередньо для оцінки складності програмного коду, для аналізу його швидкодії, ефективності та можливості адаптації у разі необхідності. Програмні метрики є важливою частиною всього процесу управління якістю ПЗ, оскільки вони можуть бути легко автоматизовані та застосовані з метою досягнення важливих технічних характеристик.

При систематичному застосуванні вони представляють собою потужний та економічно доцільний інструмент контролю якості коду, саме тому вони будуть інтегровані у розроблювану систему управління якістю. Існують різні типи метрик, кожен з яких має свою обмежену галузь застосування та складність реалізації.

До кількісних метрик відносять кількість рядків коду, аналіз функційних точок, кількість токенів та метрики Холстеда [2].

Кількість рядків коду (source lines of code – SLOC) – це метрика розміру програмного продукту, основна ідея якої полягає у підрахунку кількості рядків коду, без врахування коментарів та порожніх рядків [3]. Ця метрика розроблена для оцінки зусиль, витрачених на розробку програмного модуля та для порівняння продуктивності розробників. Метрика SLOC є однією з найпростіших і ранніх метрик, тому вона має низку обмежень. Основними з цих обмежень є те, що одна й та сама функціональність може бути як записана в один рядок, так і розбита на декілька; також потрібно враховувати специфіку окремих мов програмування. Враховуючи це, дану метрику не можна вважати гнучкою, проте вона є простою у реалізації та має широку галузь застосування. Розрізняють два типи рядків – фізичні та логічні; фізичні рядки представляють кількість усіх рядків коду, а логічні – кількість команд програми.

Кількість токенів (token count) – концепція цієї метрики полягає у тому, щоб розглядати програму як колекцію токенів. Токенами вважаються унікальні оператори та операнди, і вся логіка роботи програми може бути визначена за допомогою цих базових конструкцій [31]. Розмір програми вважається сумою кількості унікальних операторів та кількості унікальних операндів. Метрика token count має широку галузь застосування, оскільки вона може бути інтегрована у будь-який проєкт. Проте ця метрика не враховує специфіки архітектури конкретної програмної системи, а тому вона може використовуватись лише як додатковий інструмент, а не як самостійний критерій оцінки ПЗ.

Аналіз функціональних точок (function point) – виражає обсяг бізнес-логіки ПЗ за допомогою аналізу функцій та модулів системи. Усі функції програми розподіляються на п’ять типів: функції введення, функції виведення, запити, функції роботи з файлами та зовнішні інтерфейси. Після розподілу на категорії проводиться аналіз алгоритмічної складності кожної функції, на основі чого формується розмір програмного продукту [31]. Перевагою цієї метрики є її

універсальність – результати оцінки не залежать від мови програмування та архітектури додатку; проте ця метрика є доволі складною у використанні.

Метрики Холстеда (Halstead metrics) також відносяться до кількісних метрик, проте вони відображають значно більше інформації у порівнянні з попередніми метриками. Цей метод передбачає підрахунок загальної кількості операторів, загальної кількості операндів, кількості унікальних операторів та кількості унікальних операндів; за допомогою цих показників розраховуються значення розміру програми, розміру словника, складність програми, обсяг зусиль тощо [2].

Розмір словника програми η обчислюється за формулою (1):

$$\eta = \eta_1 + \eta_2 , \quad (1)$$

де η_1 – кількість унікальних операторів;

η_2 – кількість унікальних операндів.

Програмна довжина N обчислюється за формулою (2):

$$N = N_1 + N_2 , \quad (2)$$

де N_1 – загальна кількість операторів;

N_2 – загальна кількість операндів.

Очікувана програмна довжина $\widehat{N}$ обчислюється за формулою (3):

$$\widehat{N} = \eta_1 log_2 \eta_1 + \eta_2 log_2 \eta_2 , \quad (3)$$

де η_1 – кількість унікальних операторів;

η_2 – кількість унікальних операндів.

Об’єм програми V обчислюється за формулою (4):

$$V = N \times log_2 \eta, \quad (4)$$

де N – програмна довжина;

η – розмір словника програми.

Складність програми D обчислюється за формулою (5):

$$D = \frac{\eta_1}{2} \times \frac{N_2}{\eta_2}, \quad (5)$$

де N_2 – загальна кількість операндів;

η_1 – кількість унікальних операторів;

η_2 – кількість унікальних операндів.

Обсяг зусиль E обчислюється за формулою (6):

$$E = D \times V, \tag{6}$$

де D – складність програми;

V– об’єм програми.

Час роботи програми T обчислюється за формулою (7):

$$T = \frac{E}{18}, \tag{7}$$

де E – обсяг зусиль.

Прогнозоване число помилок B обчислюється за формулою (8):

$$B = \frac{E^{\frac{2}{3}}}{3000}, \tag{8}$$

де E – обсяг зусиль.

До переваг метрик Холстеда можна віднести відносну простоту використання та універсальність, оскільки ці метрики не залежать ні від мови програмування, ні від складності алгоритму, що описується. До недоліків метрик Холстеда можна віднести те, що вони є статичними, тому вони обраховуються безпосередньо за допомогою коду, і, отже, не відображають атрибути програмної системи під час її виконання.

Найпоширенішим представником метрик потоку керування програмою є цикломатична складність програми, або цикломатичне число Мак-Кейба (McCabe’s cyclomatic metric). Концепція цієї метрики полягає у представленні програми як зв’язного орієнтованого графа, вузли якого відображають частини вихідного коду, а ребра – потік керування, який виконується під час роботи програми. Цикломатична складність M рівна числу лінійно незалежних шляхів роботи програми у його відображенні за допомогою графа (9):

$$M = E - N + 2P, \tag{9}$$

де E – кількість ребер у графі;

N– кількість вершин у графі;

P– кількість компонент зв’язності графа.

Цикломатичне число описує складність роботи програми та число можливих відгалужень алгоритму роботи під час її виконання. Програмна інженерія пропонує

широкий вибір метрик програмного забезпечення які слід застосовувати опираючись на архітектуру проєкту, мову програмування та цілі, які були поставлені перед командою розробників. Розроблювана система управління якістю розробки програмних продуктів може інтегрувати в себе будь-які метрики та інші інструменти контролю якістю коду, які необхідні для використання при роботі із конкретним проєктом.

2.3 Засоби контролю якості процесів розробки

До засобів контролю якості процесів відносяться різноманітні графічні техніки та методи статистичного аналізу, що використовуються для знаходження різних, часто неочевидних, причинно-наслідкових відношень між процесами, що інтегровані у діяльність організації. Загальновідомим інструментом у сфері менеджменту проєктів є техніка оцінювання та аналізу проєкту (program evaluation and review technique – PERT). Дана техніка використовується для аналізу та оцінку ресурсів (здебільшого часових), необхідних для завершення проєкту, опираючись на його складові частини. Спочатку необхідно визначити всі завдання, необхідні для завершення проєкту, та порядок їх виконання. Після цього потрібно визначити три можливі варіанти часу виконання кожного завдання: максимальний, мінімальний та найбільш імовірний. Далі слід побудувати мережеву діаграму, в якій завдання будуть представлені мережевими вузлами, а порядок виконання – зв’язками між вузлами. Побудувавши діаграму, її можна використати для оцінки реального часу виконання проєкту - це може бути досягнуто за допомогою знаходження найдовшої послідовності виконання задач, яка називається критичним шляхом. Очікуваний час виконання задачі *te* розраховується за формулою (10):

$$te = \frac{o+4m+p}{6}, \tag{10}$$

де o – мінімальний час виконання, якщо не з’являться ніякі проблеми;

m – найбільш імовірний час виконання;

p – максимальний час виконання, якщо відбудуться усі можливі перешкоди.

Формула (11) використовується для знаходження стандартного відхилення часу виконання задачі σ_{te}:

$$\sigma_{te} = \frac{p-o}{6}, \quad (11)$$

де o – мінімальний час виконання;

p – максимальний час виконання.

Відповідно, очікуваний час виконання проєкту рівний очікуваному часу виконання задач критичного шляху TE із врахуванням стандартного відхилення σ_{TE}, описаних у формулах (12) і (13):

$$TE = \sum_{i=1}^{n} te_i, \quad (12)$$

$$\sigma_{TE} = \sqrt{\sum_{i=1}^{n} {\sigma_{te_i}}^2}, \quad (13)$$

де n – кількість задач критичного шляху;

te – очікуваний час виконання задачі;

σ_{te} – стандартне відхилення часу.

Метод PERT є доволі потужним інструментом, який допомагає не тільки визначити необхідний час виконання проєкту та всіх його підзавдань, а також правильно організувати порядок виконання паралельних задач із оптимальним використанням ресурсів.

Також слід реалізувати всі методи контролю, які давно показали свою результативність у системах управління якістю, розроблених згідно ISO 9000: причинно-наслідкову діаграму (cause-and-effect diagram), контрольний лист (check sheet), контрольну карту (control chart), діаграму Парето (Pareto chart).

Причинно-наслідкова діаграма, відома також як діаграма Ішикави, використовується для наочного відображення багатьох можливих причин проблеми. Діаграма ділиться на різні секції, в кожній з яких описується причина, а також ланцюг подій, які передували даній причині. Даний інструмент часто використовується у складних проєктах, наприклад, при проєктуванні автомобілів, оскільки даний виріб складається із багатьох різних деталей, кожна з яких може бути причиною проблеми. Дану діаграму використовують при проєктуванні та

тестуванні продукту із метою знаходження та запобігання можливих дефектів продукту шляхом аналізу усіх можливих причин.

Контрольний лист представляє собою таблицю, у яку записується розподіл частоти появи певного явища, наприклад, можна визначити день тижня, у який відбувається найбільше число спізнень. Контрольні карти, також відомі як контрольні карти Шухарта, допомагають визначити, чи є процес контрольованим – контрольна карта відображає зміну якогось показника у різні часові відрізки. Аналітик використовує наявні дані для визначення нижньої та верхньої контрольних меж процесу, на основі чого відбувається контроль.

Діаграма Парето включає в себе гістограми, що відображають розподіл якогось явища, та криву, що відображає кумулятивну залежність даного розподілу. У сфері управління якістю, діаграма Парето часто використовується для визначення пріоритетів, за якими слід усувати недоліки продукту.

Як і із засобами контролю якості програмного коду, розроблювана система управління якістю передбачає інтеграції складніших статистичних методів при необхідності, наприклад, кореляційний аналіз, регресійний аналіз, крос-табуляція, кластерний аналіз, дерева класифікації та інші. Проте описані у даному розділі методи можуть бути використані у будь-якому проєкті з метою оптимізації процесів незалежно від продукту, що розробляється.

2.4 Висновки

У другому розділі було сформовано структуру системи управління якістю розробки програмних продуктів. На основі аналізу, здійсненого у першому розділі кваліфікаційної роботи, було вирішено дотримуватись положень серії стандартів ДСТУ ISO 9000. Був здійснений опис складових частин цієї системи, були обґрунтовані рішення створення даних компонентів. Описано функції та методи системи, шляхи їх використання та реалізації. Вміст даного розділу включає опис математичних та структурних моделей розроблюваної системи. Практичну користь розроблюваної системи можна обґрунтувати дослідженнями [20, 21, 28, 33], що

визнають впровадження систем менеджменту якості суттєвим фактором підвищення ефективності роботи як невеликої команди, так і корпорації.

Шляхом інтеграції системи менеджменту якості, було удосконалено метод управління якістю розробки програмних продуктів. Набув подальшого розвитку підхід загального управління якістю. Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи опираючись на результати емпіричних досліджень [20, 21, 28, 33].

У підрозділі 2.1 було детально описано п’ять основних компонентів розроблюваної системи та їх структуру, визначено основні принципи управління якістю та методи їх дотримання, модель системи було подано на рисунках 2.1 і 2.2. Обґрунтовано потребу розробки програмного рішення, що допоможе інтегрувати систему в процес розробки програмного забезпечення.

У підрозділі 2.2 були підібрані засоби контролю якості програмного коду, що є одним із компонентів розроблюваної системи, описані всі аспекти їх використання, принцип роботи, описано обчислювальні формули.

У підрозділі 2.3 описано засоби контролю організаційних процесів, здійснено їх порівняння, описано сфери та способи застосування. Описано техніку оцінювання та аналізу термінів виконання проєкту, підібрані графічні та статистичні методи оцінки якості.

Вміст даного розділу включає опис математичних та структурних моделей розроблюваної системи.

3 АРХІТЕКТУРА ПРОГРАМНОЇ РЕАЛІЗАЦІЇ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

3.1 Формування та аналіз вимог програмної реалізації системи управління якістю

Розробка будь-якого програмного забезпечення, як найпростіших клієнтських додатків, так і складних синхронних розподілених систем, починається із етапу формування та аналізу вимог кінцевих користувачів програми.

Вимоги програмного забезпечення – це набір функцій, атрибутів та характеристик, яким програма повинна відповідати. Саме вимоги формально описують те, що система повинна виконувати, як вона повинна це виконувати, які сервіси повинна пропонувати та із якими даними працювати, архітектуру рішення тощо. Якість розробленої системи представляє собою її відповідність заданим вимогам.

Аналіз вимог полягає у їх приведенні до чітко зрозумілого вигляду, при якому вимогам притаманні наступні атрибути:

– атомарність. Вимога повинна описувати одну і лише одну річ і не може бути розбита на ряд нових вимог;

– завершеність. Кожна із вимог повинна бути повністю визначеною в одному місці, а вся необхідна для реалізації інформація повинна бути в наявності;

– послідовність. Вимога не суперечить іншим вимогам і повністю відповідає зовнішній документації;

– виконуваність. Вимога може бути реалізована в межах одного проєкту;

– недвозначність. Вимога виражає об’єктивні факти, а не суб’єктивні вподобання, можлива одна і лише одна інтерпретація;

– перевірюваність. Реалізація вимоги може бути визначена за допомогою огляду, тестування, демонстрації.

Вимоги до програмного забезпечення поділяються на дві групи: функціональні та нефункціональні. Функціональні вимоги описують те, що будуть використовувати користувачі: функції, інтерфейс, структуру тощо. Таким чином

відбувається опис поведінки інформаційної системи. Нефункціональні вимоги визначають атрибути, яким програмне забезпечення повинне відповідати, тобто це вимоги до характеру поведінки.

Отже, у результаті аналізу вимог до розроблюваної системи управління якістю розробки програмних продуктів були висунуті наступні функціональні вимоги:

– система повинна мати можливість створення, редагування та видалення різних проєктів;

– кожен проєкт повинен мати назву та опис із основною інформацією;

– система повинна мати можливість створення, редагування та видалення різних задач;

– кожна задача повинна мати назву, опис дату створення, дату початку виконання, зазначений час виконання, пріоритет та тег;

– для кожного проєкту задачі повинні бути поділеними на поточні, майбутні та виконані із можливістю перенесення задач від однієї категорії до іншої;

– повинна бути можливість використання необхідних засобів контролю якості програмного коду: огляд коду програми, автоматизований розрахунок метрик якості - метрик Холстеда та цикломатичну складність;

– повинна бути можливість використання засобів контролю якості процесів: техніку оцінки та аналізу проєктів із автоматичним обчисленням критичного шляху, використання графічних та статистичних засобів – діаграми Парето, причинно-наслідкової діаграми, аналізу за допомогою контрольного листа та контрольної карти;

– система повинна мати функцію автоматичного збереження.

Також був висунутий наступний перелік нефункціональних вимог до розроблюваної системи:

– система повинна бути надійною;

– швидкодія повинна бути високою;

– система повинна бути масштабованою;

– джерело даних повинне бути безпечним;
– інтерфейс повинен бути інтуїтивно зрозумілим;
– інтерфейс повинен бути адекватно масштабованим.

Базуючись на результатах аналізу вимог слід розробити архітектуру програмної системи із подальшою реалізацією.

3.2 Розробка архітектури програмної реалізації системи управління якістю

Безпосередньо проєктування програмної системи розпочинається із формування всіх сценаріїв використання програми та із формування схеми інформаційної системи, що включає в себе вхідні дані, основні обчислювальні процедури та результат роботи.

Кінцевими користувачами програмної системи є керівник проєкту, бізнес-аналітик та розробник програмного забезпечення. Керівник здійснює контроль поточними проєктами компанії, він створює, редагує або видаляє як основні дані щодо проєкту, так і його складові частини, тобто різні категорії задач. Керівник проєкту має можливість аналізу якості програмного коду, він може аналізувати структуру програми та виконувати обчислення програмних метрик за допомогою вбудованих функцій, може досліджувати вже проведені в системі тестування та аналізи, залучати працівників до огляду коду при наявності сумнівів щодо його якості. Також керівник може перевіряти ефективність роботи команди за допомогою інструментів контролю якості процесів, досліджувати результати статистичного аналізу, проведеного аналітиком, у вигляді візуальних діаграм та графіків. Головною відповідальністю менеджера проєкту є здійснення керування задачами, він формує план робіт, терміни виконання, відповідальних осіб, визначає пріоритети, змінює порядок виконання, визначає статус виконання тієї чи іншої задачі – система передбачає три статуси задачі: поточні, майбутні та виконані із автоматичним переходом від одного статусу до іншого, а також із можливістю ручної зміни при необхідності. Кожна задача містить лаконічний опис, який дає змогу чітко визначити межі виконання задачі та її головні елементи, звичайно,

керівник має можливість його редагування або видалення задачі, якщо виникла ситуація зміни планів внаслідок зовнішніх причин.

Розробники поділяються на тестувальників, розробників бізнес-логіки та розробників графічного інтерфейсу програми. При роботі над проєктом, вони тісно співпрацюють, оскільки тестувальники працюють із програмним кодом, написаним розробниками, а розробники здійснюють правки та рефакторинг відповідно до результатів тестувань. Кожен член команди розробників може проводити огляд коду або досліджувати результати вже проведеного огляду, оскільки він фіксується у системі, є можливість обчислення метрик розміру та складності програмного елементу для подальшої оцінки ступеня оптимізації алгоритмів та структур даних. Після проведення тестувань різних типів, тестувальник проводить аналіз отриманих результатів із формуванням подальших вказівок щодо виправлення дефектних або сумнівних компонентів програми, що будуть враховані при наступних віхах розробки програми для можуть бути зафіксовані у вимогах до розробки програмного продукту.

Головною відповідальністю аналітика у команді є здійснення контролю якості процесів, що перебігають у компанії. Аналітик досліджує всі зовнішні фактори роботи, наприклад, вплив робочого графіку, умов роботи програмістів та їх ефективність, здійснює аналіз ефективності співпраці із іншими компаніями, які постачають команду програмними модулями чи бібліотеками, досліджує рівень ефективності кожного працівника щоб визначити необхідність підвищення кваліфікації, зміни структури команди або шляху взаємодії розробників. Для цього використовуються статистичні методи оцінки та аналізу: метод PERT, причинно-наслідкові діаграми, діаграми Парето, контрольні листи та контрольні карти. Результати аналізу використовуються для прийняття рішень при менеджменті задач, плану виконання та контролю якості самого програмного коду.

Дані зв’язки зображені за допомогою діаграми варіантів використання на рисунку 3.1

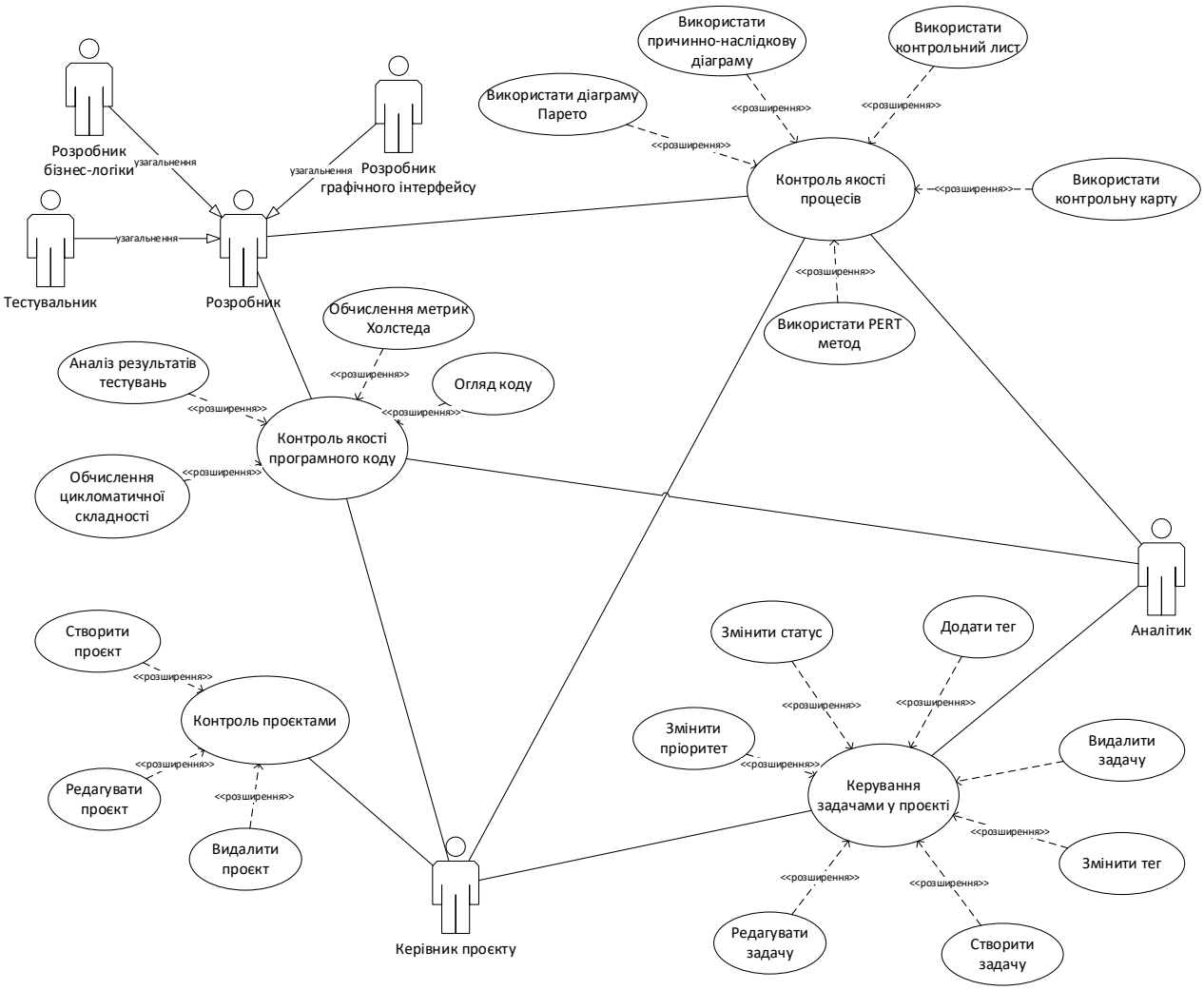


Рисунок 3.1 – Діаграма варіантів використання розроблюваної програмної реалізації

Загальний сценарій використання інформаційної системи полягає у введенні вхідної інформації у вигляді плану проєкту, структури робіт та основних їх характеристик, фіксації показників якості коду, отриманих в результаті використання різних засобів контролю та фіксації різних показників ефективності роботи, наприклад, числа спізнень, недотримання графіку, числа проведення заходів з метою підвищення кваліфікації та м’яких навичок, тощо.

Використовуючи функції інформаційної системи, здійснюється обробка вхідних даних усіма членами команди та визначаються зміни, необхідні для покращення вимірюваних показників. Дані вказівки представляють собою вихідну

інформацію із системи, яка реалізується в проєкті і процедура виконується знову. Процес управління якості є неперервним і відбувається протягом всього часу виконання проєкту. Схема інформаційної структури системи менеджменту якості представлена на рисунку 3.2.

Рисунок 3.2 – Схема інформаційної структури системи менеджменту якості

Використовуючи інформацію про вимоги до продукту, сценарії використання та інформаційну структури системи, була розроблена структурна модель компонентів розроблюваної програмної реалізації. Вона включає чотири компоненти:

- модуль управління проєктом;
- модуль контролю якості процесів;
- модуль контролю якості коду;
- джерело даних.

Джерело даних використовується для збереження інформації з усіх модулів розроблюваної системи, проте вона знаходиться поза межами даної системи, оскільки джерело даних представляє собою зовнішню сутність.

Модуль управління проєктом є головним структурним елементом моделі, оскільки він буде виконувати основні функціональні задачі: зчитувати і записувати дані у джерело даних, надавати інструменти менеджменту проєктів та інструменти менеджменту задач проєкту, обробку даних із зовнішнього сховища із метою їх передачі у інші модулі та приведення їх до відповідного формату, зрозумілого іншим компонентам, прийом оброблених даних із інших компонентів системи для збереження їх у зовнішньому сховищі.

Модуль контролю якості процесів використовуватиметься для надання статистичних методів обробки даних із модуля управління проєктом, а модуль контролю якості коду надаватиме інструмент менеджменту програмного продукту та аналізу атрибутів його якості. Модель структурних модулів представлена на рисунку 3.3.

Рисунок 3.3 – Модель структурних модулів проєктованої системи

Для кращої наочності процесів руху даних всередині системи, були розроблені діаграми потоків даних. Вони описують які дані програмна система зчитує із зовнішнього джерела даних, як ці дані рухаються між компонентами проєктованої системи та яким чином відбувається фіксація результатів аналізу та обробки у сховищі даних. Загальна діаграма потоків даних представлена на рисунку 3.4.

Рисунок 3.4 – Загальна діаграма потоків даних

Діаграма потоків даних, що представлена на рисунку 3.4 відображає лише потоки даних ззовні системи на вході і на виході, та при цьому не описує перебіг даних всередині між модулями системи. Всього визначено чотири потоки даних – один на вході системи і три на виході:

- дані про проєкти, структура задач, атрибути задач;
- результати керування проєктом;
- показники якості продукту;
- показники якості процесів.

Як було визначено раніше у даному підрозділі, функції роботи із зовнішнім джерелом даних виконуватиме модуль управління проєктом, який, в свою чергу, приводить дані до необхідного формату для використання всередині даного модуля, чи для передачі модулю контролю якості процесів або модулю контролю якості коду. При використанні функцій управління проєктом, всі дані про зміну структури та атрибутів задач залишаються всередині модуля.

При виклику функцій статистичного аналізу, до відповідного модуля відправляються дані, обрані користувачем, наприклад, дані про кількість зірваних термінів виконання робіт або дані про кількість перероблених фрагментів програмного коду за фіксований період часу, у модулі відбувається обробка цих даних, побудова відповідних діаграм та обрахунок результатів, які повертаються у модуль управління проєктом.

При використанні засобів контролю якості коду, модуль отримує дані про задачу, такі як, дата початку та час виконання, особа, що відповідальна за виконання роботи, а також інформацію про файл, у якому знаходиться програмний код. У модуль управління проєктом повертається звіт про виконані тестування та огляд коду, а також обчислені програмні метрики.

Для наочного відображення руху даних всередині розроблюваної інформаційної системи, було побудовано діаграму потоків даних між модулями програмної системи, яку зображено на рисунку 3.5.

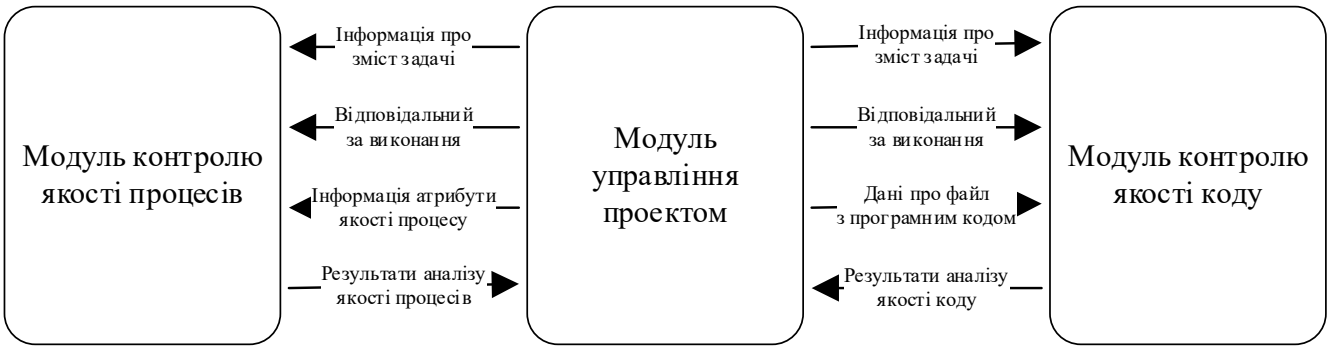


Рисунок 3.5 – Діаграма потоків даних між модулями

Як було описано вище, рисунок 3.5 показує, що модулі, відповідальні за роботу інструментів контролю якості програмного коду або процесів, отримують інформацію для роботи із модуля управляння проєктом, вона включає в себе основну інформацію про задачі, які досліджуються, а також інформацію, необхідну для використання засобів контролю якості: для побудови діаграм, обчислення програмних метрик та застосування інших засобів.

3.3 Проєктування модулів системи управління якістю

Після розробки загальної архітектури програмної реалізації системи управління якістю розробки програмних продуктів, потрібно провести детальне проєктування модулів системи та їх взаємодії між собою. Проте, перед побудовою, безпосередньо, архітектури, потрібно провести порівняльний аналіз різних поширених парадигм програмування.

Парадигма програмування – це набір понять та ідей, які визначають стиль написання програми, підхід до програмування. При детальному проєктуванні слід обов'язково визначити, згідно якої із парадигм буде проводитись розробка, оскільки це впливає на вибір технологій та мови програмування, на спроєктовану модель системи та на кінцевий вигляд програмного забезпечення. Обрана парадигма впливатиме на простоту та швидкість написання програми, на легкість її підтримки, розмір програмного коду, швидкодію застосунку на реальному апаратному забезпеченні. Сьогодні, найбільшого поширення серед розробників набули три парадигми програмування:

- процедурне програмування;
- функційне програмування;
- об'єктно-орієнтоване програмування.

Процедурне програмування описує програму як скінченну послідовність процедурних викликів. Процедури включають в себе серії обчислювальних команд, які викликаються задля отримання певного результату. Код може бути повторно використаний в різних частинах програми. Перевагою даної парадигми є простота використання та висока швидкість навчання, проте створюючи великі комплексні проєкти, використання процедурної мови програмування створює високий ризик написання надмірної кількості програмного коду, через що такий код важко читати, а програму складно підтримувати.

Функційне програмування передбачає проєктування програми як серії математичних функцій, які виступають будівельними блоками для виконування усіх задач. Функційні мови програмування уникають використання циклів та

переходів, а використовують рекурсивні функції замість цього. Ця парадигма добре підходить для рішення складних задач, проте розробка за допомогою функційних мов програмування є доволі складною, особливо у великих проєктах.

Об’єктно-орієнтоване програмування передбачає проєктування та реалізацію програми як множини класів та об’єктів – реалізацій класів. Об’єкти взаємодіють один з одним із метою отримання бажаного результату. Логіка додатку повинна бути описана всередині класу та використовуватись через створені об’єкти. Використання об’єктно-орієнтованої парадигми значно спрощує створення моделі певного явища із реального світу, оскільки будь-який реальний об’єкт можна змоделювати у вигляді класу із скінченною кількістю атрибутів, методів на зв’язків із іншими класами.

Модулі програмної реалізації системи управління якістю розробки програмних продуктів було вирішено проєктувати та розроблювати за допомогою парадигми об’єктно-орієнтованого програмування, оскільки описані раніше функціональні вимоги та варіанти використання досить просто реалізуються за допомогою системи класів, тому розробка за допомогою мови програмування даної парадигми буде простішою, а код – більш зрозумілим. Також, розроблювана система передбачає велику кількість повторного використання написаного коду, що підтверджує правильність вибору.

Перед виділенням класів для кожного програмного модуля слід описати всі структурні елементи модулів за допомогою діаграм компонентів, які описують всі елементи системи, необхідні для виконання функцій. Це допоможе чітко визначити методи та атрибути класів. Діаграма компонентів модуля управління проєктом представлена на рисунку 3.6.

Рисунок 3.6 налічує сім компонентів, необхідних для роботи модуля управлінням проєкту. Компонент роботи із базою даних представляє собою набір функцій для встановлення зв’язку із базою даних, форматування, зчитування та запису даних після обробки. Результати його роботи використовує компонент обробки проєктів, який містить функції створення, редагування та видалення проєктів, а також надає доступ до структури задач наступному компоненту –

обробки задач. Даний компонент налічує функції, аналогічні попередньому, проте він також відповідальний за роботу інструментів сортування задач за часом виконання – задачі розподіляються на поточні, виконані та майбутні. Аналогічно працює компонент сортування задач за пріоритетом, проте сортування відбувається в межах своїх категорій, оскільки поточна задача завжди має пріоритет вищий за майбутню, час виконання якої ще не прийшов. Компонент вибору задач потрібний для групування задач для подальшої обробки за допомогою статистичних інструментів.

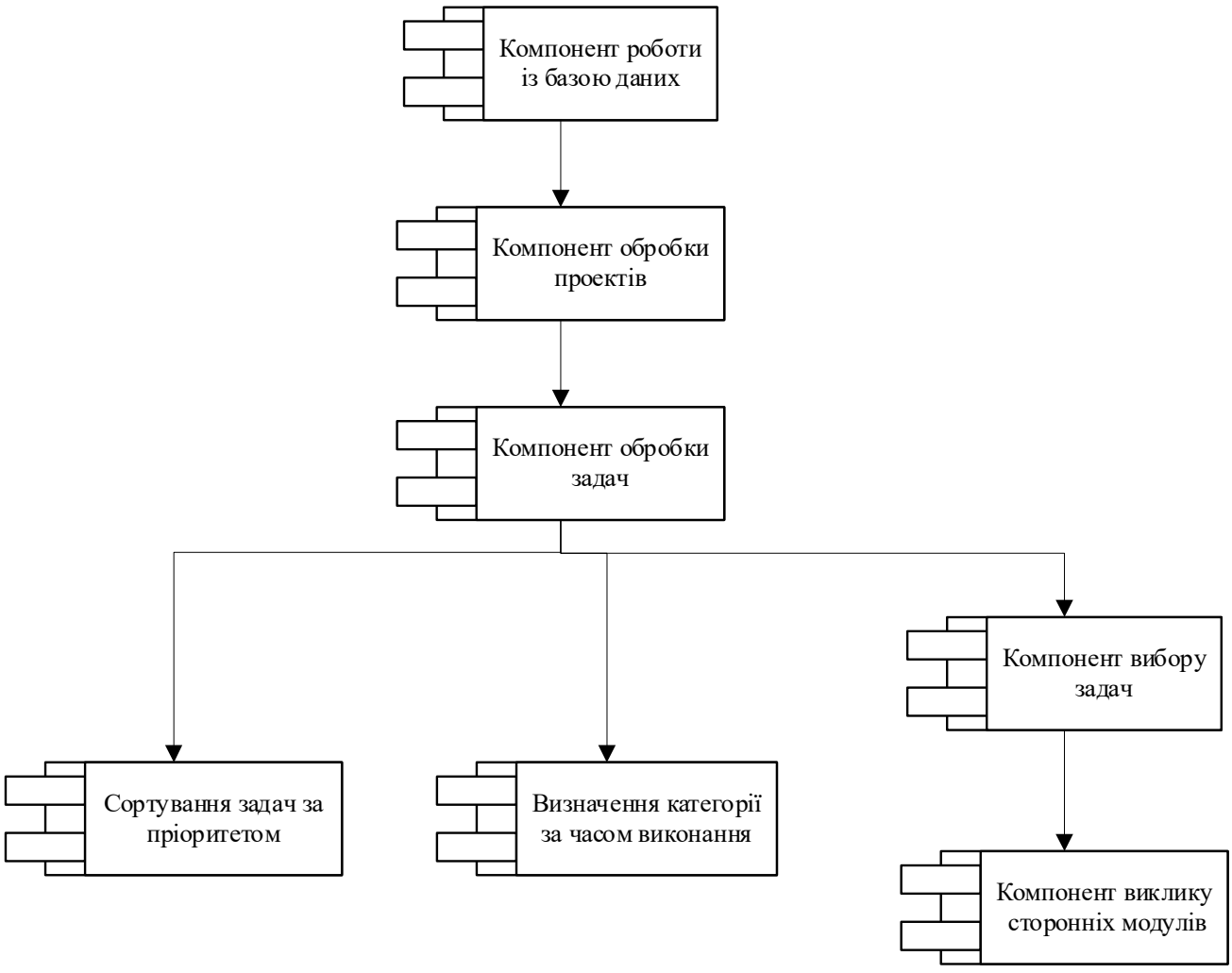


Рисунок 3.6 – Діаграма компонентів модуля управління проєктом

На рисунку 3.7 зображена діаграма компонентів модуля контролю якості коду. Базовими є компоненти обміну даними з головним модулем та компонент відкриття файлу у файловій системі локальної машини, оскільки дані компоненти

відповідальні за зчитування програмного коду для подальшої обробки. Компонент обробки потрібний для проведення ручного огляду коду та аналізу його адекватності при сумнівних значеннях метрик. Компоненти обчислення цикломатичної складності та обчислення метрик Холстеда потрібні для обчислення відповідних метрик, сумарна кількість яких дорівнює дев'ятьом. Фіксація результатів тестувань потрібна для збереження значень, отриманих ззовні програми для подальшого дослідження аналітиком.

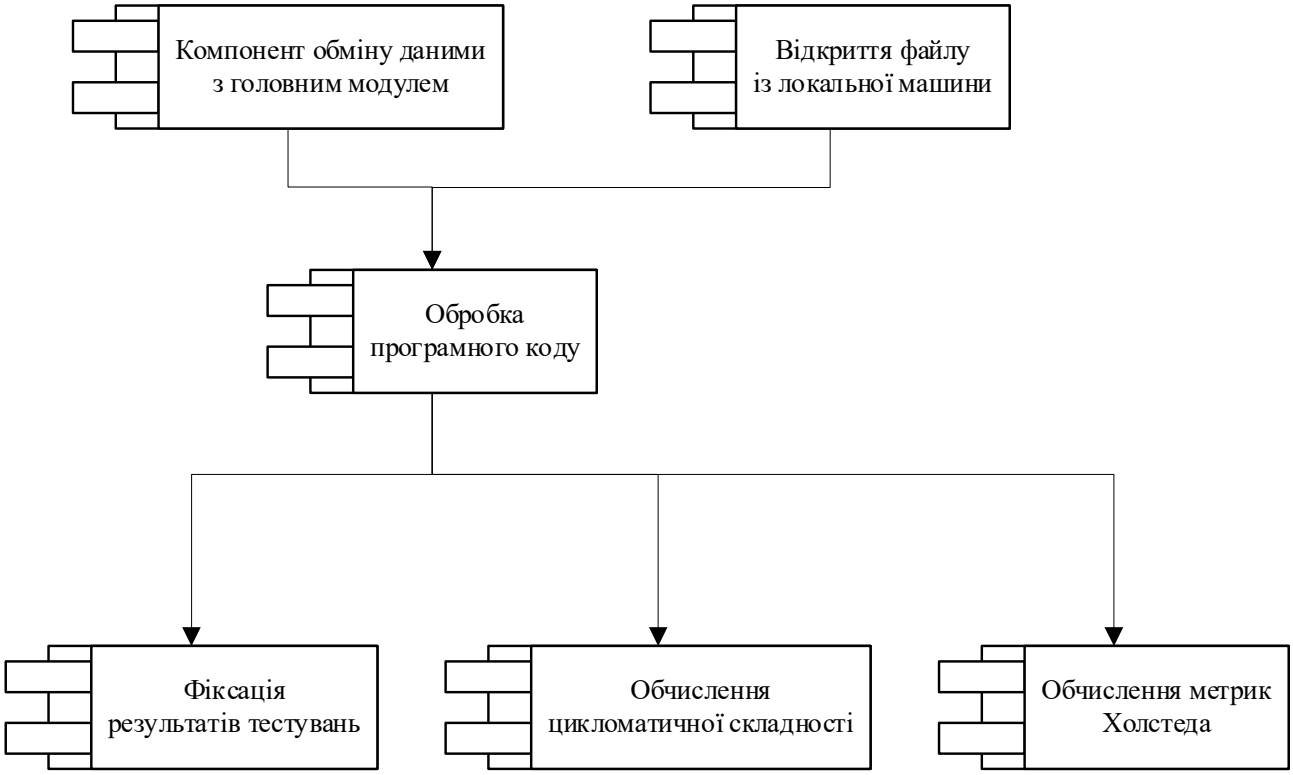


Рисунок 3.7 – Діаграма компонентів модуля контролю якості коду

Остання діаграма компонентів відображає структурні елементи модуля контролю якості процесів, вона зображена на рисунку 3.8. Як і в попередньому модулі, базовим функціональним компонентом є компонент обміну даними із головним модулем – він відповідає за зчитування даних про вибрані роботи та їх атрибути якості, які використовуються компонентом обробки даних для аналізу та компонентом розрахунку статистичних даних для визначення значень змінних, на базі яких формуються діаграми. Обчислені змінні передаються до функції візуалізації графіків або функції візуалізації діаграм відповідно до обраного

інструменту. У випадку вибору причинно-наслідкової діаграми або контрольного листа програма візуалізує діаграму, а у при виборі контрольної карти та діаграми Парето програма візуалізує графік. Хоча графік є частковим випадком діаграми, проте з точки зору візуалізації за допомогою програмних засобів, між ними є різниця у методі побудови. Створення мережевої діаграми необхідне для застосування методу PERT та обчислення найкращого, найгіршого та найбільш ймовірного часу виконання серії задач.

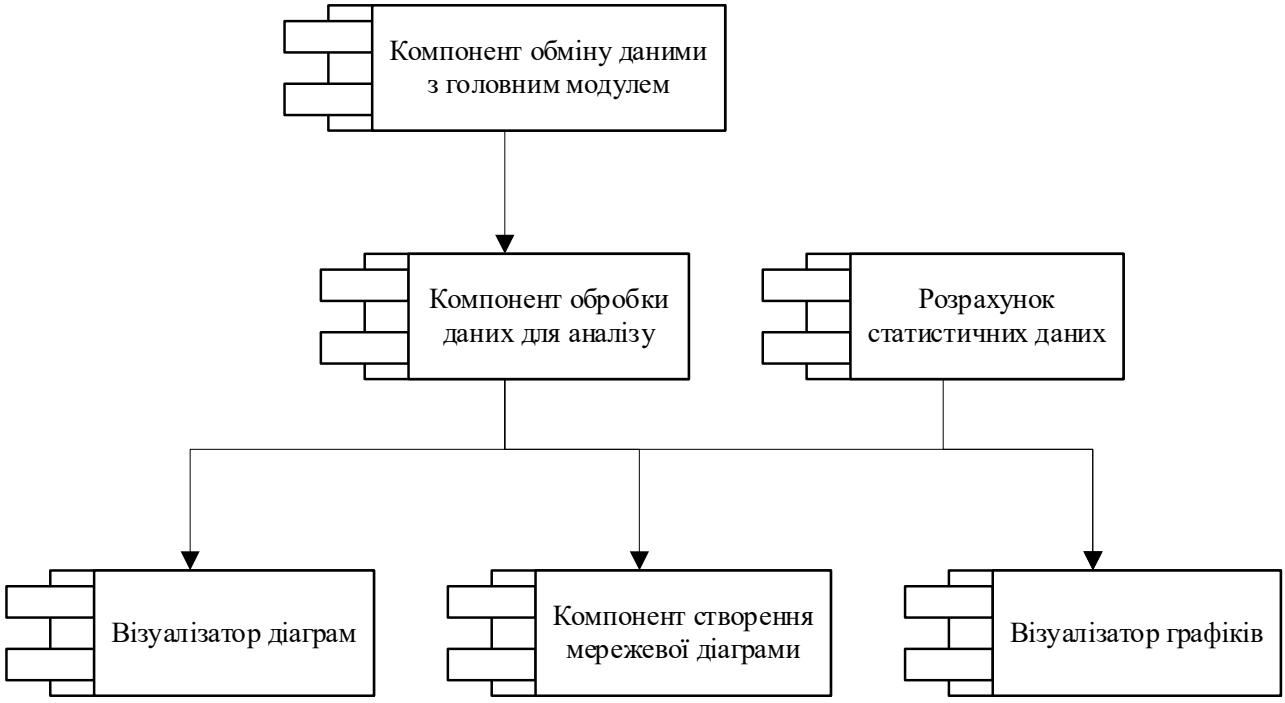


Рисунок 3.8 – Діаграма компонентів модуля контролю якості процесів

Фінальним етапом проєктування інформаційної системи згідно парадигми об’єктно-орієнтованого програмування є побудова діаграми класів. Вона складається із усіх класів та взаємозв’язків між ними, кожен клас включає в себе методи та атрибути, які будуть використовуватись у програмній реалізації.

Для виділення класів з метою побудови діаграми слід використовувати результати попереднього аналізу, тобто діаграми компонентів та діаграми потоків даних. Побудована діаграма класів розроблюваної інформаційної системи наведена на рисунку 3.9.

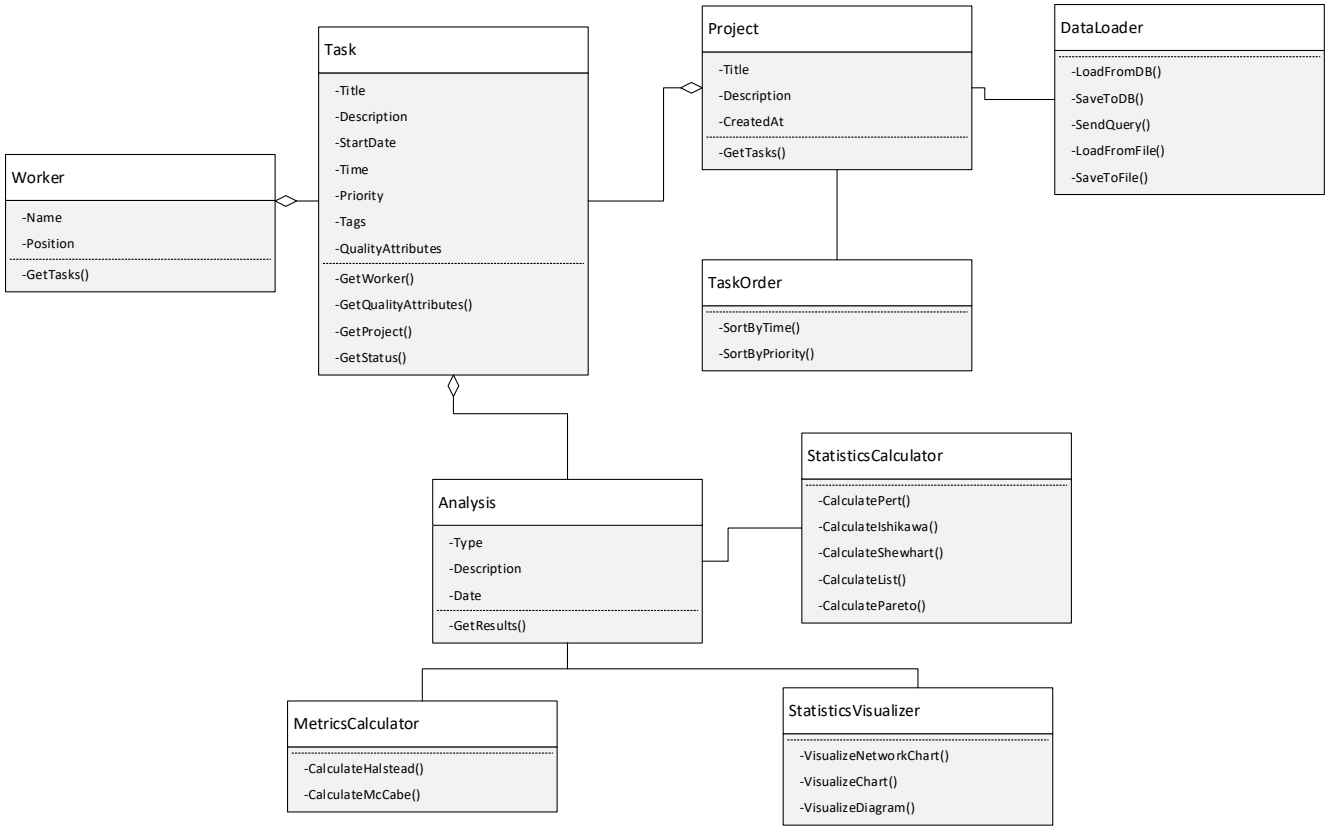


Рисунок 3.9 – Діаграма класів інформаційної системи

Діаграма налічує дев’ять класів системи, що відповідають за основну логіку роботи частини описаних вище компонентів, наприклад, компонент вибору задач, буде реалізований як елемент графічного інтерфейсу програми.

Клас Project вміщує поля, необхідні для збереження інформації про проєкти, і метод GetTasks(), який повертає всі задачі, які відносяться до обраного проєкту. Перебуває у відношенні агрегації із класом Task, оскільки один проєкт вміщує декілька задач, а також у відношеннях асоціації із класами TaskOrder і DataLoader. Клас TaskOrder містить методи сортування задач: SortByTime() потрібний для сортування задач між категоріями, тобто відносить їх до поточних, майбутніх або виконаних, метод SortByPriority() виконує сортування в межах часової категорії за пріоритетом, який може бути високим, середнім, низьким, або може бути відсутнім. DataLoader потрібний для виконання операцій роботи із базою даних або файловою системою комп’ютера. Методи LoadFromDB() та SaveToDB() відповідають за завантаження та збереження даних про об’єкти відповідно, методи LoadFromFile() та SaveToFile() потрібні для виконання операцій зчитування та запису інформації у

файл. Метод SendQuery() потрібний для того, щоб відправити SQL-запит до бази даних за необхідності.

Клас Task зберігає у своїх полях інформацію про задачі проєкту: назву, опис, дату початку, кількість днів на виконання, пріоритет, тег, атрибути якості виконання. Перебуває у відношенні агрегації із класами Worker і Analysis, оскільки в одного працівника є декілька робіт, а в для однієї задачі можна провести декілька аналізів. Методи класу призначені для отримання інформації про пов’язані об’єкти, наприклад, метод GetQualityAttributes() повертає атрибути якості при їх наявності, до показників ефективності виконаної роботи можна віднести: запізнення початку виконання через зірвані попередні терміни або навпаки, виконання роботи раніше визначеного терміну внаслідок неправильної оцінки керівником складності завдання та рівня кваліфікації працівника. Клас Worker, пов’язаний зв’язком агрегації, містить всього два поля, які зберігають інформацію про ім’я працівника та його посаду у компанії, метод GetTasks() повертає множину задач працівника, що містяться інформаційній системі.

Клас Analysis містить інформацію про заходи з метою оцінки ефективності, що відбуваються в компанії. Поле Type зберігає інформацію про тип – може бути використання засобів контролю якості коду або використання засобів контролю якості процесів, поле Description містить опис основних аспектів проведеного аналізу, поле Date зберігає дату проведення аналізу. Метод GetResults() повертає інформацію про результати проведеного аналізу: опис проведеного аналізу, визначені результати, обчислені змінні. Клас Analysis перебуває у відношенні асоціації із трьома класами: StatisticsCalculator, StatisticsVisualizer, MetricsCalculator. Клас StatisticsCalculator потрібний для обчислення статистичних змінних та містить методи для виконання різних типів аналізу: метод CalculatePert() обчислює змінні техніки оцінювання та аналізу програм, необхідні для визначення критичного шляху виконання проєкту, метод CalculateIshikawa() встановлює зв’язки та відношення у причинно-наслідковій діаграмі, метод CalculateShewhart() обчислює верхні та нижню контрольні лінії контрольної карти, метод CalculateList() обчислює показники контрольного листа і метод CalculatePareto()

обчислює дані для побудови кумулятивної кривої та гістограм для подальшої їх візуалізації. Клас StatisticsVisualizer містить три методи для візуалізації засобів контролю якості процесів: VisualizeNetworkChart() потрібний для побудови да модифікації мережевого графіка, методи VisualizeChart() та VisualizeDiagram() виконують візуалізацію графіків та діаграм відповідно. Клас MetricsCalculator ключає два методи для обрахунку програмних метрик: CalculateHalstead() обчислює вісім метрик розмірності та складності програми, метод CalculateMcCabe() обчислює число цикломатичної складності програми.

Після опису роботи програми за допомогою наведених раніше діаграм та після побудови діаграми класів потрібно обрати мову програмування та інші прикладні інструменти розробки, що дозволить виконати програмну реалізації розробленої системи управління якістю.

3.4 Висновки

У даному розділі було виконано опис архітектури програмної реалізації системи управління якістю розробки програмних продуктів. Це було здійснено у декілька етапів:

- формування та аналіз вимог;
- визначення сценаріїв функціонування програми та побудова діаграми варіантів використання;
- побудова загальної схеми інформаційної структури розроблюваної системи та виділення основних модулів;
- визначення потоків руху інформації та побудова діаграм потоків даних;
- вибір парадигми програмування;
- побудова детальних діаграм компонентів модулів програмної системи;
- визначення структури класів та побудова діаграми класів.

У підрозділі 3.1 здійснено опис переліку функціональних та нефункціональних вимог до програмної реалізації.

У підрозділі 3.2 виділено типи основних користувачів програми, їх відповідальності та потреби, описано функції, необхідні для роботи кожного типу користувача. Виділено зовнішні та внутрішні модулі роботи системи, описано процеси перебігу даних між системою та джерелом даних і всередині модулів розроблюваної системи, на основі чого побудовано діаграми потоків даних.

У підрозділі 3.3 здійснено порівняння поширених парадигм програмування та обґрунтовано вибір об’єктно-орієнтованої парадигми. Виконано детальний поділ трьох модулів програми на компоненти функціонування та побудовано діаграми компонентів. Вкінці підрозділу здійснено опис основних класів об’єктно-орієнтованої системи та відображено їх структуру на діаграмі класів.

У третьому розділі описано загальний алгоритм роботи користувача із програмою та здійснено розробку архітектури та детальної моделі інформаційної системи.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ УПРАВЛІННЯ ЯКІСТЮ РОЗРОБКИ ПРОГРАМНИХ ПРОДУКТІВ

4.1 Вибір інструментарію та середовища реалізації

Початковим кроком розробки будь-якого програмного забезпечення є обґрунтування вибору мови програмування, середовища розробки та бібліотек, що будуть використовуватись під час розробки. Оскільки система спроєктована у об’єктно-орієнтованій парадигмі і представляє собою множину класів, то і мову програмування слід обирати серед тих, що підтримають розробку в об’єктно-орієнтованій парадигмі. Найпопулярнішими мовами програмування з таких є C++, C#, Java та Python.

C++ – це мова програмування високого рівня, яка була розроблена у 1985 році. Оскільки C++ бере за основу синтаксис логіку побудови програм та компілятор мови C, програми, розроблені на C++, відрізняються високою потужністю та швидкодією на реальній машині, порівняно з іншими мовами програмування. Хоча дана мова підходить для розробки об’єктно-орієнтованих додатків, проте основною сферою її використання є розробка програм із низькорівневою роботою із апаратним забезпеченням. Усі із зазначених певною мірою беруть свій початок з C++. Дана мова програмування стоїть біля витоків об’єктно-орієнтованого підходу і все ще використовується, але її популярність впала з появою більш простих і надійних технологій. Адже по-перше, C++ на сьогоднішній день є доволі складною у використанні та вивченні мовою програмування, яка до того ж не підтримує автоматичну збірку сміття. Це вимагає від програміста належного рівня підготовки і навичок, розуміння механізмів роботи з вказівниками, тощо. Відзначається, що один з найбільших недоліків сьогодення є дуже повільний розвиток мови з рідкісним виходом нових стандартів. C++ потребує більше часу для розробки, а також ринок пропонує не надто багато бібліотек та інструментів для побудови сучасного графічного інтерфейсу.

C# – високорівнева мова програмування, що була розроблена у 2000 році. На відміну від C++, програміст не має потреби концентруватись над такими

проблемами, як керування потоками та виділенням пам’яті для об’єктів. Гарно підходить для розробки сучасних асинхронних додатків за різними шаблонами. C# був розроблений компанією Microsoft як аналог Java. Мультиплатформність, збирач сміття, своєрідна форма байт-коду, і строга типізація так само присутні. На відміну від Java, C# розвивається набагато активніше і в багатьох напрямках. Популярними платформами на базі C# є WPF (Windows Presentation Foundation) та ASP.NET. WPF – це платформа розробки користувацького інтерфейсу для створення додатків для настільних систем. Підтримує широкий набір компонентів розробки програм, включаючи ресурси, елементи керування, графіку тощо. ASP.NET – це інтернет-платформа для створення веб-сайтів або систем із клієнт-серверною архітектурою за допомогою HTML, CSS і JavaScript. Включає в себе багато готових елементів для створенні інтерактивних динамічних сторінок. В цілому, індустрія пропонує багато сторонніх бібліотек та пакетів, що допомагають полегшити розробку комплексних додатків на C#.

Java – це об’єктно-орієнтована мова програмування, розроблена у 1995 році. Також має широку підтримку розробників, безліч допоміжних сторонніх компонентів розробки та використовується у багатьох сферах програмування, основними з яких є розробка мобільних додатків та бізнес-логіки серверної частини ресурсів, що функціонують у глобальній мережі. Головною перевагою Java є можливість створення крос-платформних застосунків, оскільки будь-який Java програмний код запускається у спеціальній віртуальній машині Java Virtual Machine (JVM). Використання мови Java у програмі гарантує високий ступінь масштабованості на різних системах із різними архітектурами як операційної системи, так і апаратних модулів. Проте, порівняно із C#, Java пропонує складніший і не такий інтуїтивно зрозумілий синтаксис, також є потреба у детальному налаштуванні та підтримці найбільш нової версії залежностей продукту.

Python – це мова програмування високого рівня із динамічною типізацією, що була розроблена у 1991 році. Перевагою Python є велика кількість готових модулів та бібліотек для проведення статистичного аналізу, а також простий

синтаксис, додатки, написані на Python, як правило, відрізняються меншим об’ємом коду та розміром програми в цілому. Інакшою перевагою є використання динамічної типізації, що дає можливість написання більш лаконічних програм. До недоліків можна віднести те, що команди у Python виконуються за допомогою інтерпретатора, що знижує загальну швидкодію додатків, призначених для активного використання користувачем. Також, Python гірше підходить для розробки графічного інтерфейсу, порівняно із іншими мовами, і передбачається використання інших мов програмування для розробки та підтримки для графічної частини програми.

У результаті проведеного раніше порівняльного аналізу об’єктно-орієнтованих мов програмування було вирішено, що найбільш оптимальним рішенням буде використання мови C# та написання програми для персональних комп’ютерів використовуючи платформу Windows Presentation Form, оскільки це дає змогу розробки сучасного графічного інтерфейсу за допомогою зручних у використанні інструментів. Також, використання даної платформи спрощує процес роботи із базою даних, так як інтегровані середовища розробки дають змогу підключення локальної бази даних на персональному комп’ютері без необхідності використання стороннього програмного забезпечення. Дана мова буде використовуватись для розробки усіх модулів інформаційної системи.

Для полегшення розробки вирішено використовувати пакет бібліотек MindFusion.WPF, який пропонує набір інструментів для побудови різних графіків та діаграм, включаючи всі необхідні у даному проєкті, із прямою інтеграцією у розмітку програмних вікон, розроблений на платформі WPF.

Оскільки розробка буде вестись на C#, то найбільш раціональним вибором інтегрованого середовища розробки є Microsoft Visual Studio, оскільки він містить зручний текстовий редактор із вбудованою функцією автоматичного підбору під назвою Microsoft IntelliSense, потужні засоби відлагоджування програм, інструменти візуалізації вікон графічного інтерфейсу в процесі їх розробки та вбудований менеджер пакетів NuGet, через який можна безпосередньо завантажити та інтегрувати сторонні бібліотеки C# в операційну систему.

Windows Presentation Foundation передбачає побудову додатків для персональних комп’ютерів на базі архітектурного шаблону model-view-viewmodel (MVVM), яка полягає у побудові програми за допомогою трьох компонентів:

- модель (model) ;
- представлення (view);
- модель представлення (view of model).

Модель представляє собою клас, який вміщує всю логіку додатку, обчислювальні функції, збереження інформації в змінних, визначає структури даних додатку незалежно від компоненти, відповідальної за графічний інтерфейс користувача.

Представлення, як і в класичному шаблоні model-view-controller (MVC) – це компонент системи, який є розміткою вікна графічного інтерфейсу користувача: всі поля, кнопки, вікна, спадаючі списки тощо. У WPF розмітка інтерфейсу виконується за допомогою мови XAML (eXtensible Application Markup Language).

Модель представлення – це абстракція представлення за допомогою використання класів, описуючи всі анімації та інтерактивні елементи у методах та атрибутах. Саме через модель представлення відбувається зв’язок між логікою додатку та графічним інтерфейсом користувача, а також логіка роботи самого інтерфейсу.

4.2 Програмна реалізація та тестування системи управління якістю розробки програмних продуктів

Використовуючи результати дослідження, проведеного в попередніх розділах, тобто обрану парадигму, мову та інструменти розробки, спроєктовану модель класів програмної системи та інформаційну схему руху потоків даних, було розроблено програмну реалізацію системи управління якістю розробки програмних продуктів.

На рисунку 4.1 подане стартове вікно вибору проєкту.

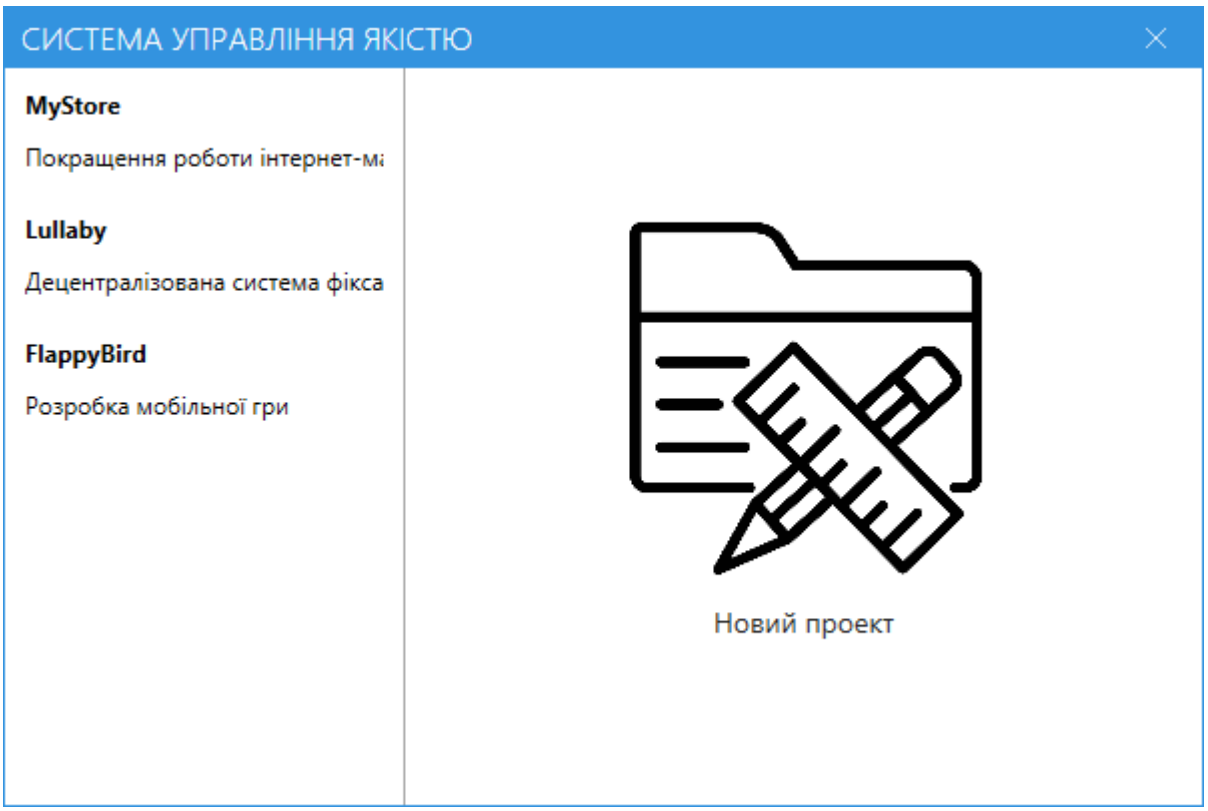


Рисунок 4.1 – Вікно вибору проєкту

Використовуючи елементи керування даного вікна, користувач може обрати робочий проєкт, видалити проєкт або створити новий, увівши при цьому його назву та короткий опис.

Як і передбачено архітектурним шаблоном MVVM у WPF, було розроблено представлення, модель та модель представлення. Фрагмент коду розмітки представлення MainWindow.xaml для функціонування вікна вибору проєкту наведений нижче:

```
<Grid>
 <Grid.ColumnDefinitions>
   <ColumnDefinition Width="{Binding ListBoxSource.Count,
 Converter={Resource       Width_Converter}}" />
 </Grid.ColumnDefinitions>
 <ListBox Grid.Column="0"
 ItemsSource="{Binding ListBoxSource}"
 SelectedItem="{Binding SelectedBoard}"
 BorderThickness="0 0 1 0"
 BorderBrush="DarkGray"
 HorizontalContentAlignment="Stretch"
 ScrollViewer.HorizontalScrollBarVisibility="Disabled"
 ScrollViewer.VerticalScrollBarVisibility="Disabled">
```

Логіка роботи графічного інтерфейсу описана у класі MainWindowViewModel, який наслідує клас ViewModelBase, необхідний для використання при побудові моделей представлень за допомогою інструментів платформи WPF. Інформація про проєкти зберігається використовуючи модель проєкту, яка описана за допомогою класу Project.cs, фрагмент програмного коду якого наведений нижче:

```
public class Project
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public DateTime CreatedAt { get; } = DateTime.Now;
        public ObservableCollection<Task> Tasks { get; set; } = new
ObservableCollection<Task>();
    }
```

Після вибору потрібного проєкту, користувач переходить до вікна задач проєкту, зображеного на рисунку 4.2.

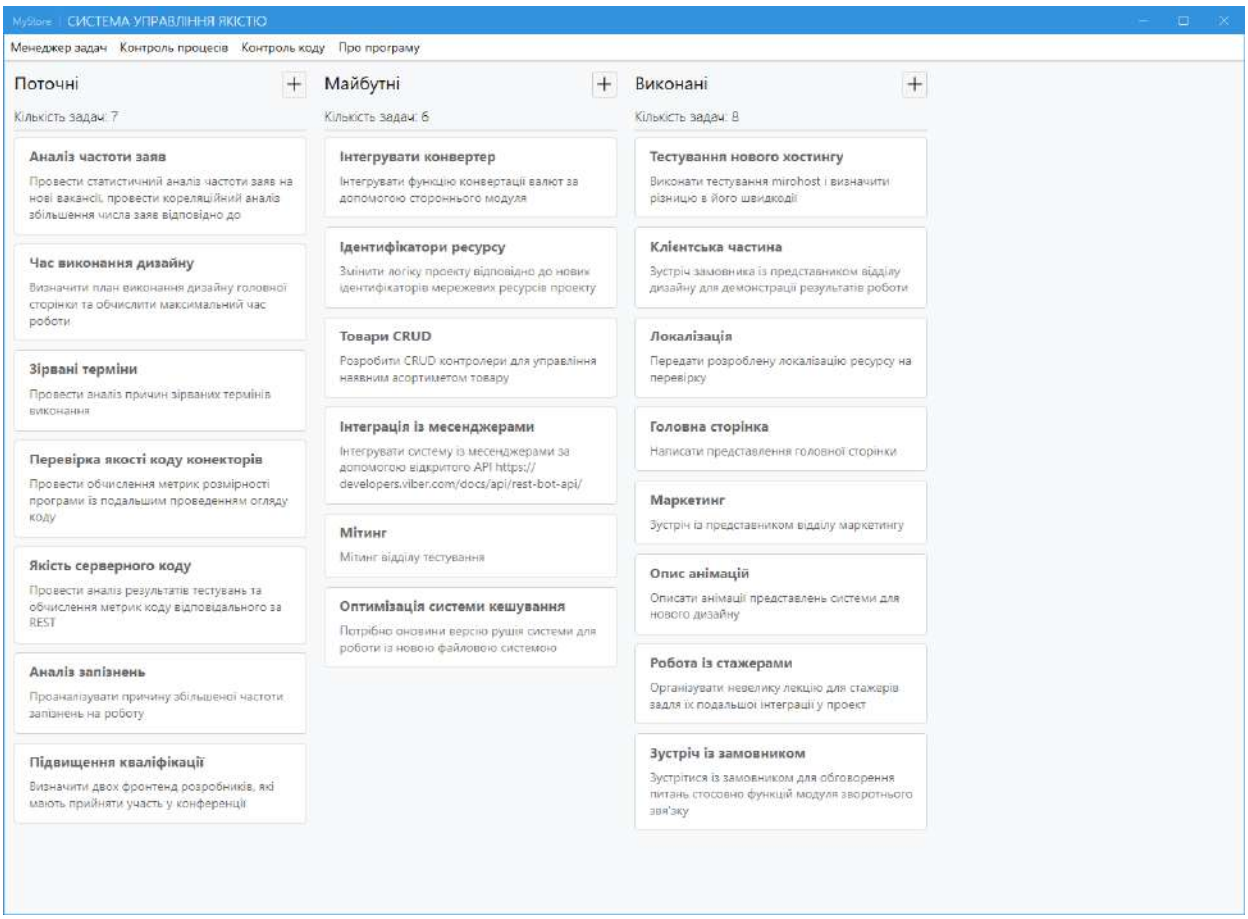


Рисунок 4.2 – Вікно керування задачами

Вікно поділене на різні частини із підписами статусу виконання задачі, яких є три: поточні, майбутні і виконані, під назвами міститься число задач із зазначеним статусом. Верхня панель інструментів вікна пропонує виклик інтерфейсу інструментів контролю коду, інструментів контролю процесів та виклик довідки про програму.

Для кожної задачі виділений фрагмент вікна, на якому відображені назва задачі та її короткий опис. Задачі можна перетягувати у інший стовпчик, тримаючи ліву кнопку миші, при цьому змінюється і статус задачі. Якщо натиснути на блок задачі, відкриється панель модифікації задачі, яка подана на рисунку 4.3.

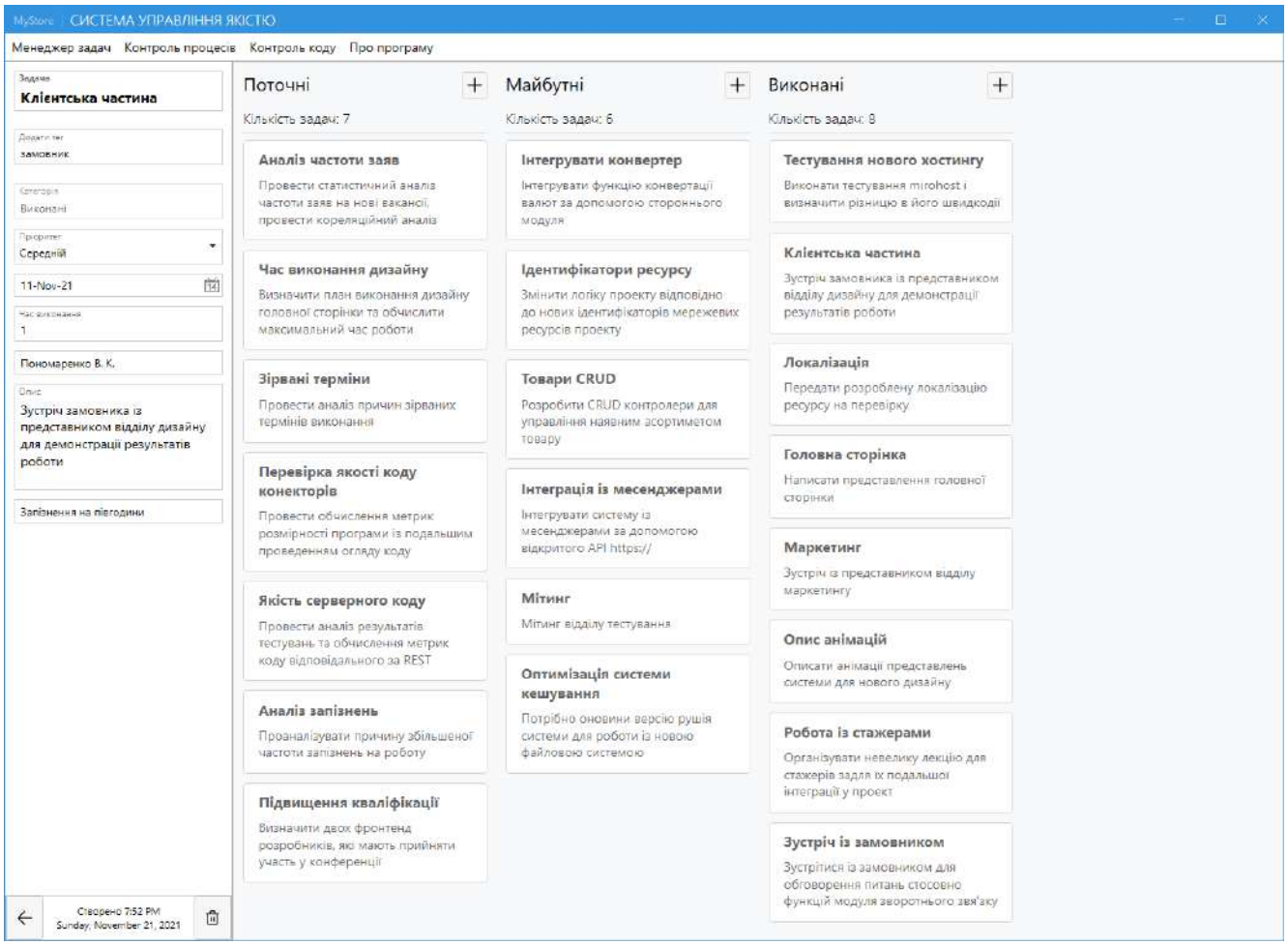


Рисунок 4.3 – Панель модифікації задачі

Дана панель містить поля, які необхідно заповнити для функціонування відповідного класу, вона налічує такі поля для введення: назва задачі, тег, статус виконання, пріоритет, дата початку виконання, кількість днів для виконання задачі,

ім’я особи, відповідальної за результати роботи, опис суті задачі, атрибут якості. Визначення пріоритету реалізоване за допомогою спадаючого списку, який пропонує три значення: високий, середній та низький пріоритет виконання задачі. Поле атрибуту якості потрібне для фіксації показників якості виконання задля подальшої обробки статистичними методами – сюди фіксуються як позитивні, так і негативні фактори, які не були передбачені раніше і через які план виконання робіт був порушений, наприклад, запізнення на роботу з якоїсь причини, або зірвані терміни виконання як результат поганого планування керівника або недостатнього рівня кваліфікації робітника.

Нижня частина панелі вміщує кнопки для видалення задачі, закриття панелі, а також дату та час створення задачі в системі.

Як було описано, дане вікно є головним у програмній системі оскільки із його панелі інструментів здійснюється доступ до засобів контролю якості. Вибір того чи інакшого інструменту здійснюється за допомогою спадаючого списку. Вікно побудови PERT діаграми надане на рисунку 4.4.

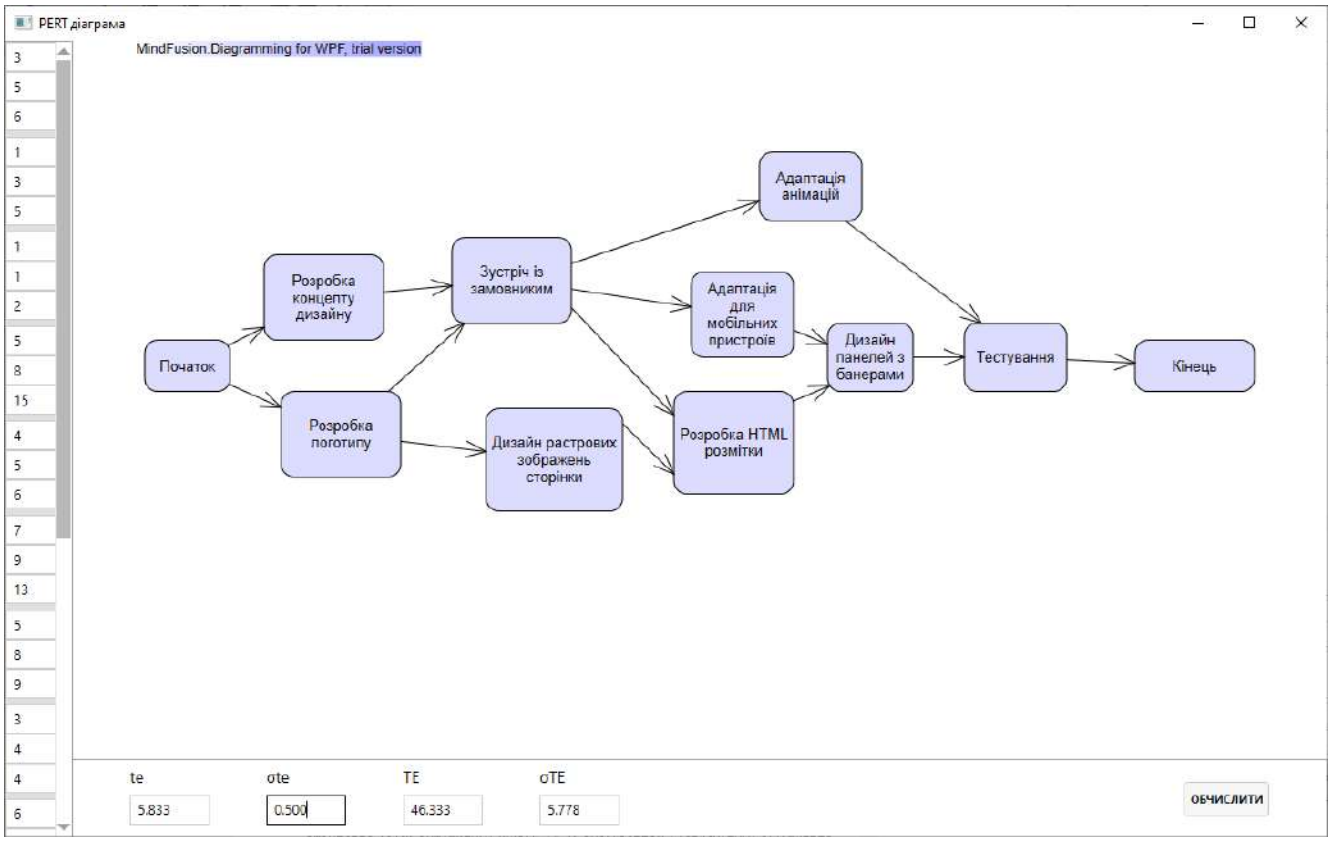


Рисунок 4.4 – Побудова PERT діаграми

При додаванні нових робіт на діаграму, користувач записує у полях зліва значення максимального, мінімального та очікуваного часу виконання роботи. Після закінчення побудови структури робіт, обчислюється довжина критичного шляху – найбільш ймовірного часу виконання всього проєкту, та значення можливого відхилення.

На рисунку 4.5 зображена причинно-наслідкова діаграма.

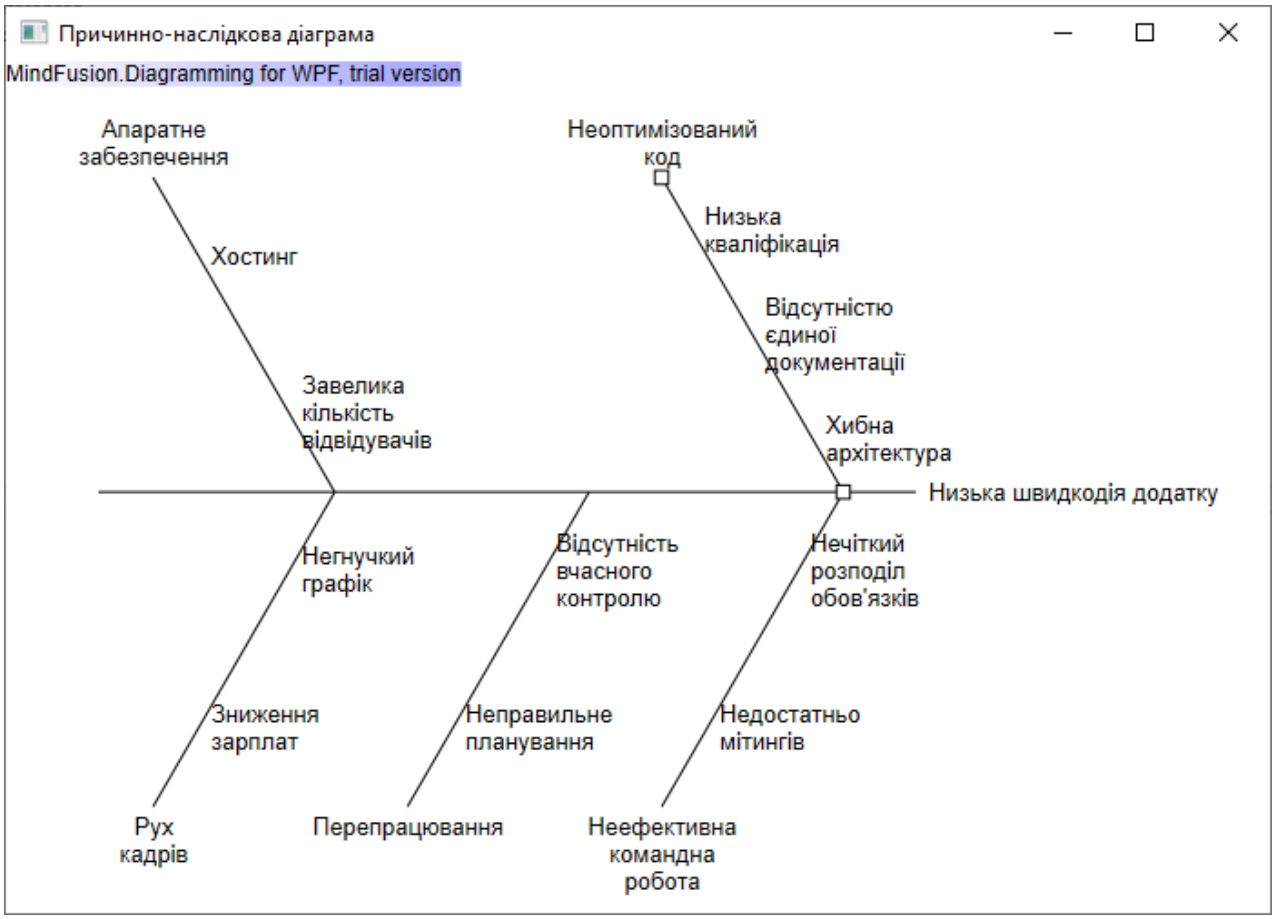


Рисунок 4.5 – Причинно-наслідкова діаграма

Як було описано у другому розділі, дана діаграма використовується для визначення причин появи того чи іншого небажаного фактору. Вона може бути сформована із полів атрибутів якості процесу, які відповідають тій чи іншій задачі у системі.

Контрольні карти Шухарта використовуються для визначення того, чи є процес контрольованим. Для цього вираховуються значення верхньої та нижньої контрольних меж процесу, в межах якої процес вважається контрольованим, далі

наносяться точки за якими будується графік для наочного відображення процесу. Використання контрольної карти надане на рисунку 4.6.

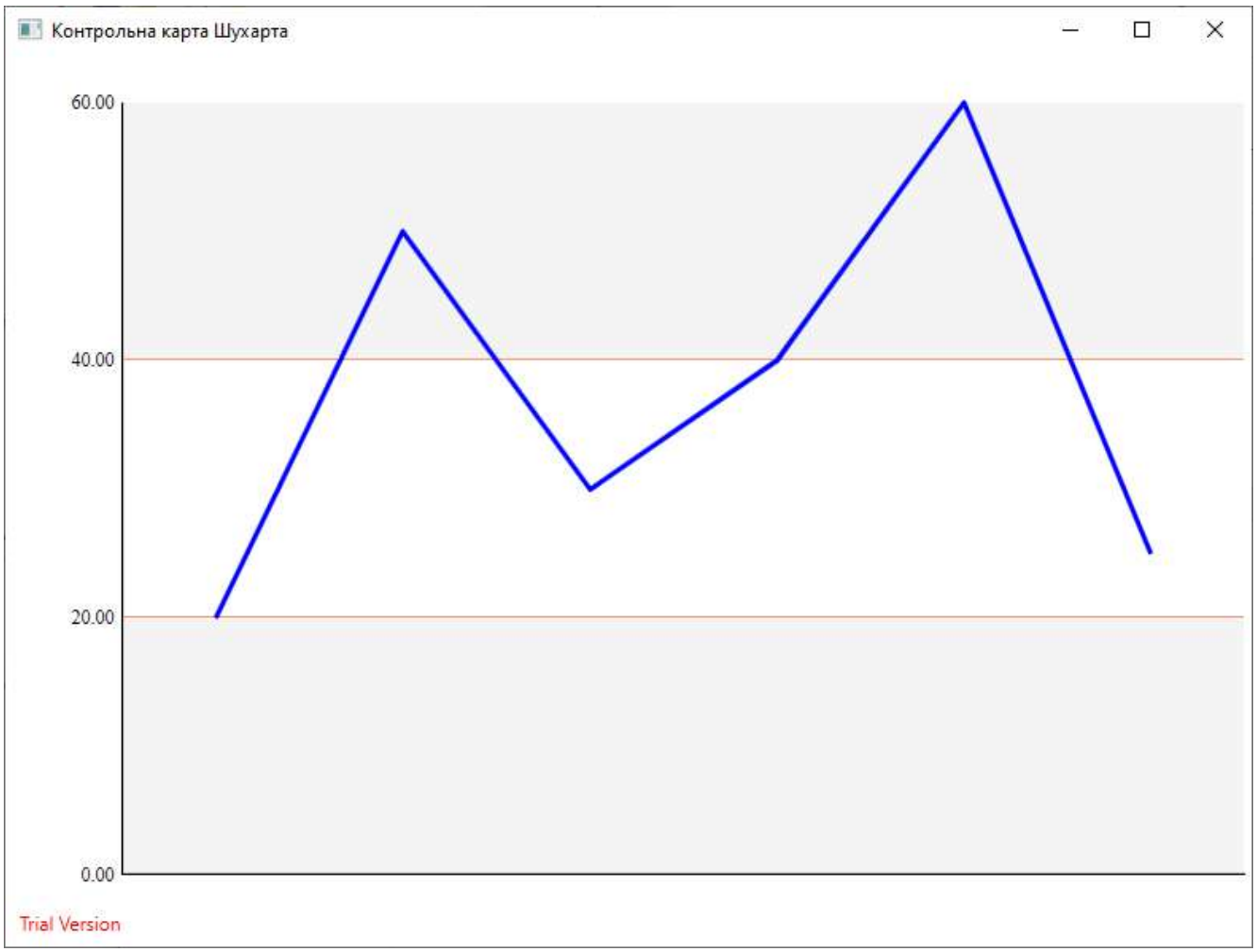


Рисунок 4.6 – Контрольна карта Шухарта

Останнім із запропонованих статистичних засобів управління якістю процесу є діаграма Парето. Вона представляє набір гістограм розподілу причин прояву певного досліджуваного явища та кумулятивну криву даного розподілу.

Даний інструмент використовується для аналізу будь-якого явища, наприклад, на рисунку 4.7 надана діаграма Парето, яка зображує розподіл факторів, що спричинили запізнення працівників на роботу, які включають наступні причини: віддаленість офісу, навантажений дорожній рух, погодні умови, сімейні обставини, пересипання через ненормований графік роботи тощо. Рисунок 4.7 надано нижче.

Рисунок 4.7 – Діаграма Парето

Одним із нововведень розроблюваної системи є те, що для аналітиків та керівників є доступними засоби контролю якості коду. У межах даного програмного продукту, засоби визначені оглядом коду, фіксацією результатів тестувань та обчисленням програмних метрик.

Для огляду коду можна завантажити файл із бази даних або відкрити її у файловому менеджері операційної системи персонального комп’ютера. Аналізуючи якість коду, результати можна фіксувати у спеціально відведеному вікно справа від основного інтерфейсу. Також тут тестувальники можуть фіксувати результати своєї роботи у вигляді, доступному для керівника, аналітиків та інших робітників, які не працюють безпосередньо з кодом.

Обчислення метрик Холстеда та метрики цикломатичної складності програмного коду відбувається автоматично із подальшим збереженням результатів. Фрагмент коду, поданий нижче потрібний для обчислення метрик:

```
class HalsteadMetrics
{
    public double calculateDifficulty()
      {
      Difficulty = (DistinctiveOperators / 2) * (TotalOperands /
    DistinctiveOperands);
      Console.WriteLine ("Difficulty= " + Difficulty);
      return Difficulty;
      }
 }
```

Вікно інтерфейсу засобів контролю якості коду представлено на рисунку 4.8.

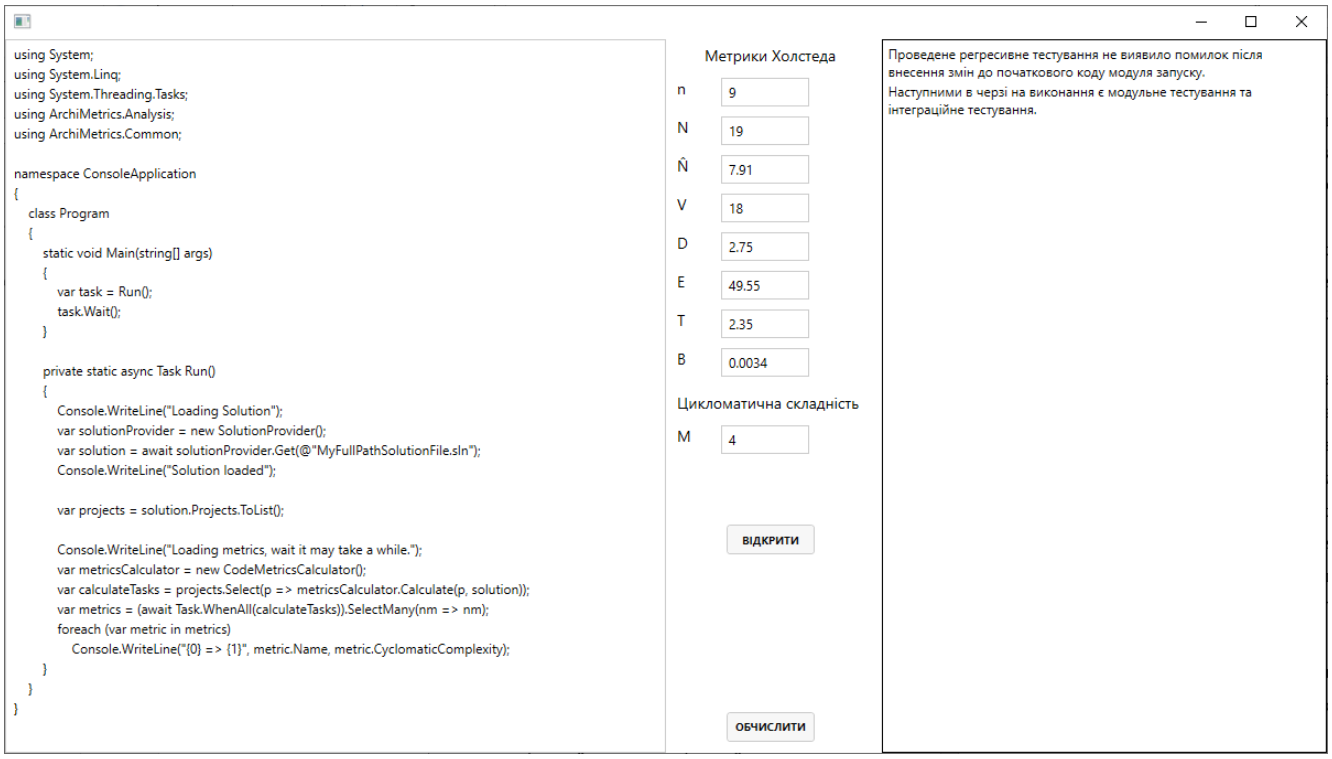


Рисунок 4.8 – Засоби контролю якості програмного коду

Також у програмі створене вікно довідки про програму, яке викликається із панелі інструментів головного вікна програми. Вигляд даного вікна наданий на рисунку 4.9.

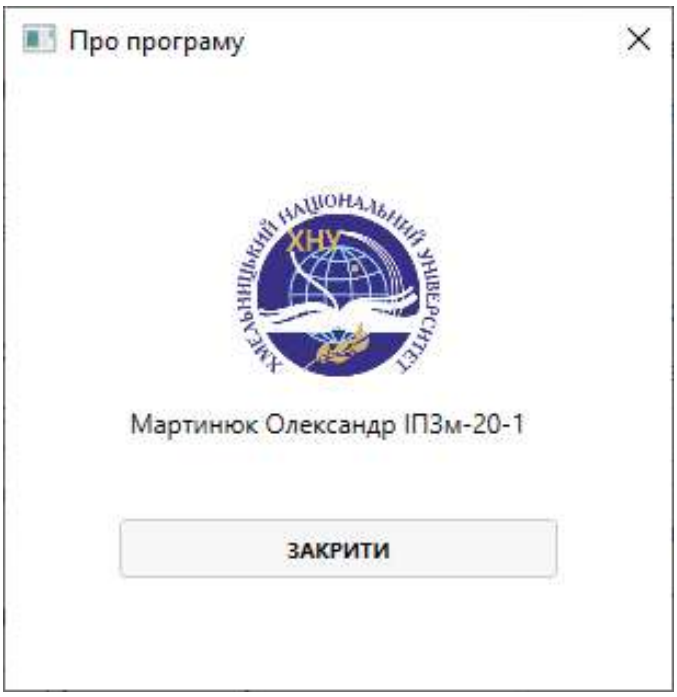


Рисунок 4.9 – Довідка про програму

На рисунках даного підрозділу представлено результати роботи функціональних компонентів програмної реалізації системи управління якістю розробки програмних продуктів. Окрім основного переліку методів, у програмі реалізовані різні механізми уникнення помилок при неправильних діях користувача. Дані механізми є частиною користувацького інтерфейсу програми і не пов’язані із логічною структурою класів додатку. Наприклад, інтерфейс програми блокує реєстрацію нової задачі, якщо вона має пусте поле назви, що представлено на рисунку 4.10.

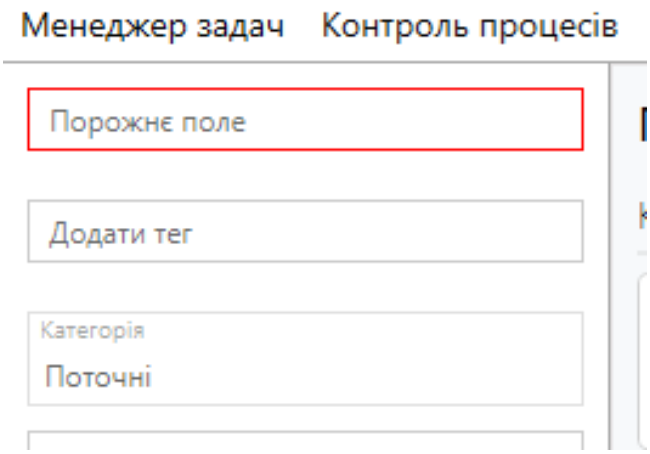


Рисунок 4.10 – Не заповнене поле при реєстрації задачі

На рисунку 4.11 зображене діалогове вікно, яке повідомляє про помилку при запуску одного із інструментів статистичного аналізу користувачем.

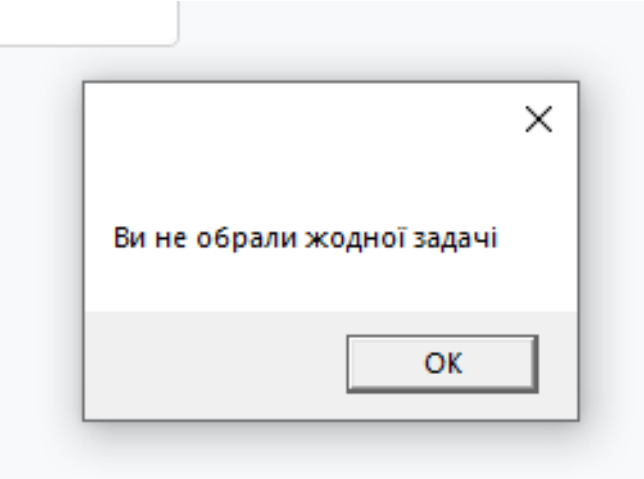


Рисунок 4.11 – Не заповнене поле при реєстрації задачі

Хоча функціонал системи може бути розширений шляхом інтеграції додаткових методів контролю якості коду або контролю якості процесів, розроблений програмний продукт є доволі ефективним із практичної точки зору, оскільки він дає можливість застосування системи управління якістю розробки програмних продуктів у реальних проєктах.

4.3 Висновки

У розділі описано програмну реалізацію системи управління якістю розробки програмних продуктів, проведено опис інструментів, вікон інтерфейсу програми, здійснено тестування програми на предмет запобігання неправильному введенню даних.

У підрозділі 3.1 було проведено порівняльний аналіз поширених мов програмування для розробки програмного забезпечення у об’єктно-орієнтованій парадигмі. Були описані основні переваги та недоліки кожної з них, в результаті чого був обґрунтований вибір мови програмування C# для розробки інформаційної системи. Було проведено опис та порівняння платформ розробки, які базуються на

C#, в результаті чого було обрано Windows Presentation Foundation як платформу для розробки програми для персонального комп’ютера. Після цього був здійснений вибір та детальний опис архітектурного шаблону model-view-viewmodel. Був зазначений пакет бібліотек від сторонніх розробників, що використовувався у програмі, та інтегроване середовище розробки.

У підрозділі 3.2 був здійснений опис структури додатка із точки зору кінцевого користувача, було продемонстровано усі основні робочі вікна графічного інтерфейсу програми. Був проведений детальний опис усіх елементів графічного інтерфейсу та пояснені всі сценарії використання програмного забезпечення користувачами. Були продемонстровані фрагменти програмного коду додатку, наведений приклад частини реалізованої у вигляді класу на мові C# моделі, частини реалізованого у вигляді розмітки на мові XAML представлення графічного інтерфейсу робочого вікна та продемонстровано зразок обчислювальної процедури у вигляді методу класу.

Наукова новизна проведеного дослідження:

– отримав подальший розвиток підхід загального управління якістю у напрямку його застосування у сфері розробки програмних продуктів, що дозволяє підвищити якість програмного коду та процесів його розробки;

– удосконалено систему менеджменту якості TQM у сфері розробки програмних продуктів, що поєднує як засоби аналізу якості коду, так і методи статистичного аналізу якості процесів;

– удосконалено метод управління якості розробки програмних продуктів.

Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи опираючись на результати емпіричних досліджень [20, 21, 28, 33].

Практичне значення отриманих результатів полягає у використанні запропонованого підходу для компаній-розробників програмного забезпечення з метою впровадження системи менеджменту якості, що відповідає стандартам ISO 9000.

Запропоновану стратегію в подальшому можна розвивати шляхом інтеграції нових засобів контролю якості, нових метрик, типів тестувань, методів статистичного аналізу. Також, можна продовжити ідеї загального управління якістю і поширити їх на нові аспекти галузі розробки програмного забезпечення, такі як способи взаємодії працівників, організацію навколишнього середовища, методи підбору робочого графіку програмістів.

ВИСНОВКИ

На основі результатів досліджень, проведених у ході кваліфікаційної роботи, було розроблено систему управління якістю розробки програмних продуктів.

У першому розділі кваліфікаційної роботи було проведено дослідження літератури та існуючих рішень з метою проведення порівняльного аналізу моделей розробки програмного забезпечення, порівняння систем менеджменту якості та визначення методів покращення якості коду та процесів розробки. В результаті було прийнято рішення про вдосконалення системи менеджменту якості розробки програмних продуктів, яка усуватиме недоліки моделей розробки програмного забезпечення.

У другому розділі було сформовано структурні компоненти системи управління якістю розробки програмного забезпечення на базі TQM, виділено основні її принципи та регламентовано використання методів контролю якості коду та методів контролю якості процесів.

У третьому розділі було виконане проєктування програмної реалізації системи менеджменту якості, що включає в себе побудову інформаційної структури розроблюваної системи, визначення парадигми програмування та побудову детального проєкту системи.

У четвертому розділі здійснено вибір мови програмування, засобів та інструментів розробки, за допомогою чого було розроблено та детально описано програмну реалізацію системи управління якості програмних продуктів, проведено її тестування та визначено практичну значимість результатів.

Поставлена мета кваліфікаційного дослідження полягала в удосконаленні методу управління якістю розробки програмних продуктів. Дане дослідження концентрувалось на вирішенні проблеми відсутності контролю ефективності проведення процесів розробки, а рішення базувалось на вирішенні даної проблеми шляхом впровадження системи менеджменту якості розробки програмних продуктів.

Наукова новизна отриманих результатів:

– отримав подальший розвиток підхід загального управління якістю у напрямку його застосування у сфері розробки програмних продуктів, що дозволяє підвищити якість програмного коду та процесів його розробки;

– удосконалено систему менеджменту якості TQM у сфері розробки програмних продуктів, що поєднує як засоби аналізу якості коду, так і методи статистичного аналізу якості процесів;

– удосконалено метод управління якості розробки програмних продуктів.

Практичне значення отриманих результатів полягає у вдосконаленні методу управління якістю розробки програмних продуктів, шляхом впровадження отриманої системи менеджменту якості у процес розробки, що дає змогу суттєво покращити ефективність роботи команди шляхом підвищення якості коду та оптимізації організаційних процесів.

Запропонований метод можна удосконалювати шляхом інтеграції нових засобів контролю якості, нових метрик, типів тестувань, методів статистичного аналізу.

За результатами досліджень була опублікована наукова стаття «Порівняння метрик оцінки якості програмних продуктів» та були опубліковані тези «Система управління якістю у розробці програмних продуктів» на конференції «Актуальні проблеми комп’ютерних наук АПКН-2021».

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Daniel Galin. Software quality. Concepts and practice / Wiley-IEEE Press, 2018. – 697 p.

2. Howard Garston-Smith. Software quality assurance. A guide for developers and auditors/ CRC Press, 1997. – 455 p.

3. Witold Suryn. Software quality engineering. A practitioner's approach / Wiley-IEEE Press, 2014. – 193 p.

4. Software Development Life Cycle Models-A Comparative Study [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/346819120_Software_Development_Life_Cycle_Models-A_Comparative_Study

5. A comprative study of sdlc model [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/305863548_A_comprative_study_of_sdlc_model

6. Prototyping-Oriented Software Development - Concepts and Tools [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/2464810_Prototyping-Oriented_Software_Development_-_Concepts_and_Tools

7. PROTOTYPING VS. SPECIFYING: A MULTI-PROJECT EXPERIMENT [Електронний ресурс]. - Режим доступу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.365.9704&rep=rep1&type=pdf

8. A Spiral Software Engineering Model to Inspire Innovation and Creativity of University Students [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/337460467_A_Spiral_Software_Engineering_Model_to_Inspire_Innovation_and_Creativity_of_University_Students

9. A Comparison Between Three SDLC Models Waterfall Model, Spiral Model, and Incremental/Iterative Model [Електронний ресурс]. - Режим доступу:

https://sijal-technology.com/images/company/86b122d4358357d834a87ce618a55de0.pdf

10. Організація роботи служби охорони праці на підприємстві [Електронний ресурс]. - Режим доступу: http://te.dsp.gov.ua/organizatsiya-roboty-sluzhby-ohorony-pratsi-na-pidpryyemstvi/

11. Object-Oriented Development Process and Metrics [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/325746397_Object-Oriented_Development_Process_and_Metrics

12. Incremental Development [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/topics/computer-science/incremental-development

13. Software Evolution and the Staged Model of the Software Lifecycle [Електронний ресурс]. - Режим доступу: https://www.sciencedirect.com/science/article/abs/pii/S0065245802800031

14. Lyssa Adkins. Coaching Agile Teams: A Companion for ScrumMasters, Agile Coaches, and Project Managers in Transition/Addison-Wesley Professional, 2010. – 352 p.

15. Comparative Analysis of Software Development Methods between Parallel, V-Shaped and Iterative [Електронний ресурс]. - Режим доступу: https://arxiv.org/ftp/arxiv/papers/1710/1710.07014.pdf

16. Quality management system [Електронний ресурс]. - Режим доступу: http://surl.li/aumvy

17. Nancy R. Tague. Quality Toolbox/ ASQ Quality Press, 2015. – 584 p.

18. Poornima M. Charantimath. Total Quality Management/ Pearson Education, 2017. – 640 p.

19. Загальне управління якістю [Електронний ресурс]. - Режим доступу: http://surl.li/aumvt

20. The impact of Total Quality Management on organizational performance [Електронний ресурс]. - Режим доступу:

https://www.researchgate.net/publication/294886200_The_impact_of_Total_Quality_Management_on_organizational_performance

21. Total Quality Management Practices' Effects on Quality Performance and Innovative Performance [Електронний ресурс]. - Режим доступу: http://surl.li/aumvo

22. ISO 9000 [Електронний ресурс]. - Режим доступу: http://surl.li/aumvw

23. ISO 9000:2015 [Електронний ресурс]. - Режим доступу: https://www.iso.org/obp/ui/#iso:std:iso:9000:ed-4:v1:ru

24. ДСТУ ISO 9000:2015 [Електронний ресурс]. - Режим доступу: https://khoda.gov.ua/image/catalog/files/%209000.pdf

25. An Empirical Study of the ISO 9000 Standards' Contribution Towards Total quality management [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/27379869_An_Empirical_Study_of_the_ISO_9000_Standards'_Contribution_Towards_Total_quality_management

26. Функціонування і розвиток системи управління якістю : навч.-метод. матеріали / В. М. Сороко. – К. : НАДУ, 2013. – 80 с

27. What is lean manufacturing? [Електронний ресурс]. - Режим доступу: https://searcherp.techtarget.com/definition/lean-production

28. An Empirical Research on Lean Production Awareness [Електронний ресурс]. - Режим доступу: https://link.springer.com/article/10.1007/s42943-020-00010-8

29. Daniel Galin. Software quality. Concepts and practice / Wiley-IEEE Press, 2018. – 697 p.

30. Howard Garston-Smith. Software quality assurance. A guide for developers and auditors/ CRC Press, 1997. – 455 p.

31. Analysis of Software Quality Using Software Metric [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/328830202_Analysis_of_Software_Quality_Using_Software_Metrics

32. Типове положення про спеціальне навчання, інструктажі та перевірку знань з питань пожежної безпеки на підприємствах, в установах та організаціях України [Електронний ресурс]. - Режим доступу: https://zakon.rada.gov.ua/laws/show/z0308-94#Text

33. Motivation and Efficiency of Quality Management Systems Implementation: A Study of Lithuanian Organizations [Електронний ресурс]. - Режим доступу: https://www.researchgate.net/publication/264857878_Motivation_and_Efficiency_of_Quality_Management_Systems_Implementation_A_Study_of_Lithuanian_Organizations

ДОДАТОК А

(обов’язковий)

ПРОГРАМНИЙ КОД ОСНОВНИХ МОДУЛІВ

А.1 Основні моделі розробленої системи

```
public class Project
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public DateTime CreatedAt { get; } = DateTime.Now;
        public ObservableCollection<Task> Tasks { get; set; } = new
ObservableCollection<Task>();
    }


    public class Task
    {
        public int Id { get; set; }
        public int ProjectId { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Category { get; set; }
        public string Priority { get; set; }
        public DateTime CreatedAt { get; set; }
        public DateTime Start { get; set; }
        public int dayscount { get; set; }
        public ObservableCollection<Tag> Tags { get; set; }
        public ObservableCollection<SubTask> SubTasks { get; set; }
    }


class Analysis
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Type { get; set; }
        public string GetResults { get; set; }

          }
```

А.2 Основні моделі представлення розробленої системи

```
public class BoardViewModel : ViewModelBase
    {
        public BoardWindow BoardWindow { get; set; }
```

```
private Board _board;

public Board Board
{
    get => _board;
    set
    {
        _board = value;
        OnPropertyChanged();
    }
}

private Task _selectedTask = new Task();

public Task SelectedTask
{
    get => _selectedTask;
    set
    {
        _selectedTask = value;
        OnPropertyChanged();
    }
}

private SubTask _newSubTask = new SubTask();

public SubTask NewSubTask
{
    get => _newSubTask;
    set
    {
        _newSubTask = value;
        OnPropertyChanged();
    }
}

private int _taskViewWidth;

public int TaskViewWidth
{
    get => _taskViewWidth;
    set
    {
        _taskViewWidth = value;
        OnPropertyChanged();
    }
}

public ICommand CloseBoardCommand { get; }
public ICommand AddTagCommand { get; }
public ICommand AddSubTaskCommand { get; }
public ICommand DeleteSubTaskCommand { get; }
public ICommand DeleteTagCommand { get; }
public ICommand DeleteTaskCommand { get; }
public ICommand CloseTaskViewCommand { get; }
public ICommand AddTaskCommand { get; }
public ICommand SaveBoardCommand { get; }

public BoardViewModel()
{
```

```
            CloseBoardCommand = new RelayCommand(CloseBoardShowMain, o => true);
            AddTaskCommand = new RelayCommand(AddTask, o => true);
            AddTagCommand = new RelayCommand(AddNewTag, o =>
!string.IsNullOrWhiteSpace(o.ToString()));
            AddSubTaskCommand = new RelayCommand(AddNewSubTask, o =>
!string.IsNullOrWhiteSpace(_newSubTask.Title));
            SaveBoardCommand = new RelayCommand(SaveBoard, o => true);
            DeleteSubTaskCommand = new RelayCommand(o =>
SelectedTask.SubTasks.Remove((SubTask) o), o => true);
            DeleteTagCommand = new RelayCommand(o => SelectedTask.Tags.Remove((Tag) o), o
=> true);
            DeleteTaskCommand = new RelayCommand(RemoveSelectedTask, o => SelectedTask !=
null);
            CloseTaskViewCommand = new RelayCommand(o => TaskViewWidth = 0, o => true);
        }

        private void CloseBoardShowMain(object parameter)
        {
            SaveBoard(parameter);
            App.WindowService.ShowMainCloseBoard();
        }

        private void RemoveSelectedTask(object parameter)
        {
            Board.Tasks.Remove(SelectedTask);
            TaskViewWidth = 0;
        }

        private void AddNewSubTask(object parameter)
        {
            SelectedTask.SubTasks.Add(new SubTask {Completed = NewSubTask.Completed, Title
= NewSubTask.Title});
                    }

        private void AddNewTag(object parameter)
        {
            SelectedTask.Tags.Add(new Tag {Name = parameter.ToString()});
            BoardWindow.ClearTaskTagEntry();
        }

        private void AddTask(object parameter)
        {
            var task = new Task
            {
                BoardId = Board.Id, Category = (string) parameter, CreatedAt =
DateTime.Now,
                Description = "", Title = "", SubTasks = new
ObservableCollection<SubTask>(),
                Priority = "None", Tags = new ObservableCollection<Tag>()
            };

            Board.Tasks.Insert(0, task);
            SelectedTask = task;
            TaskViewWidth = 250;
        }

        public void SaveBoard(object parameter)
        {
            App.UnitOfWork.Boards.Update(Board);
        }
```

```
    }
      }
```

А.3 Основні представлення розробленої системи

```
<mah:MetroWindow.Resources>
        <converters:SubTasksToStringConverter x:Key="ToStringConverter" />
        <converters:StringToColorConverter x:Key="StringToColorConverter" />
        <converters:FromNumberToVisibilityConverter
x:Key="FromNumberToVisibilityConverter" />
    </mah:MetroWindow.Resources>
    <mah:MetroWindow.LeftWindowCommands>
        <mah:WindowCommands>
            <Button Content="{Binding Board.Name}" />
        </mah:WindowCommands>
    </mah:MetroWindow.LeftWindowCommands>
   <DockPanel>
        <Menu DockPanel.Dock="Top">
            <MenuItem Header="Менеджер задач">

            </MenuItem>
            <MenuItem Header="Контроль процесів">
                <MenuItem Header="PERT діаграма"
                          Click="DiagramMenuItem_Click"
                           />
                <MenuItem Header="Причинно-наслідкова діаграма"
                          Click="CauseEffectMenuItem_Click"
                           />
                <MenuItem Header="Контрольний лист" />
                <MenuItem Header="Контрольна карта"
                           />
                <MenuItem Header="Діаграма Парето" />
               </MenuItem>
            <MenuItem Header="Контроль коду">
                <MenuItem Header="Програмні метрики"
                          Click="MetricsMenuItem_Click"
                           />
           </MenuItem>
            <MenuItem Header="Про програму"
                      Click="AboutMenuItem_Click" >
            </MenuItem>
      </Menu>
       <Border DockPanel.Dock="Left"
                BorderThickness="0 1 1 0"
                BorderBrush="DarkGray"
                Width="{Binding TaskViewWidth}">
            <DockPanel>
                <Border DockPanel.Dock="Bottom"
                        BorderThickness="1"
                        BorderBrush="DarkGray"
                        Height="50">
                    <Grid>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="2*" />
                            <ColumnDefinition Width="8*" />
                            <ColumnDefinition Width="2*" />
                        </Grid.ColumnDefinitions>
                        <Button Grid.Column="0"
```

```
                                Command="{Binding CloseTaskViewCommand}">
                            <Button.ContentTemplate>
                                <DataTemplate>
                                    <StackPanel>
                                        <icon:FeatherIcons Kind="ArrowLeft" />
                                    </StackPanel>
                                </DataTemplate>
                            </Button.ContentTemplate>
                        </Button>
                        <StackPanel
                            Grid.Column="1"
="Опис"
                                    MinLines="3" />

                            <TextBox
                                    Margin="0 0 0 10"

                                    mah:TextBoxHelper.Watermark="Атрибут якості"/>

                            <ListBox ScrollViewer.HorizontalScrollBarVisibility="Disabled"
                                    ScrollViewer.VerticalScrollBarVisibility="Disabled"
/>
                        </StackPanel>
                    </StackPanel>
                </ScrollViewer>
            </DockPanel>
        </Border>
        <Border BorderBrush="DarkGray" BorderThickness="0 1 0 0">
            <syncfusion:SfKanban MinColumnWidth="200"
                                 MaxColumnWidth="400"
                                 Margin="0"
                                 AutoGenerateColumns="False"
                                 ColumnMappingPath="Category"
                                 CardTapped="SfKanban_OnCardTapped"
                                 ItemsSource="{Binding Board.Tasks}">
                <syncfusion:SfKanban.ColumnHeaderTemplate>
                    <DataTemplate DataType="syncfusion:ColumnTag">
                        <Border>
                            <StackPanel Orientation="Vertical"
                                        Margin="0 10 0 0">
                                <DockPanel>
                                    <Button DockPanel.Dock="Right"
                                            Command="{Binding
RelativeSource={RelativeSource AncestorType={x:Type syncfusion:SfKanban}},
Path=DataContext.AddTaskCommand}"
                                            CommandParameter="{Binding Header}">
                                        <Button.ContentTemplate>
                                            <DataTemplate>
                                                <StackPanel>
                                                    <icon:FeatherIcons Kind="Plus" />
                                                </StackPanel>
                                            </DataTemplate>
                                        </Button.ContentTemplate>
                                    </Button>
                                    <TextBlock Text="{Binding Header}"
                                               FontSize="20" />
                                </DockPanel>
                                <TextBlock Text="{Binding CardCount,
StringFormat=Кількість задач: {0}}"
                                           FontSize="16"
```

```
                                        Margin="0 10 0 0"
                                        FontWeight="Thin" />
                            </StackPanel>
                        </Border>
                    </DataTemplate>
                </syncfusion:SfKanban.ColumnHeaderTemplate>
                <syncfusion:SfKanban.CardTemplate>
                    <DataTemplate>
                        <Border BorderBrush="{Binding Priority, Converter={StaticResource
StringToColorConverter}}"
                                BorderThickness="1"
                                CornerRadius="4"
                                Background="White"
                                Margin="0,5,0,5" d:DataContext="{d:DesignInstance
entities:Task}">
                                        Visibility="{Binding SubTasks.Count,
Converter={StaticResource FromNumberToVisibilityConverter}}" />
                                </StackPanel>
                                <TextBox FontSize="14"
                                         BorderThickness="0"
                                         IsEnabled="False"
                                         HorizontalAlignment="Left"
                                         Text="{Binding Description}"
                                         MaxLines="3"
                                         TextWrapping="Wrap" />
                                <ItemsControl ItemsSource="{Binding Tags}">
                                    <ItemsControl.ItemsPanel>
                                        <ItemsPanelTemplate>
                                            <WrapPanel />
                                        </ItemsPanelTemplate>
                                    </ItemsControl.ItemsPanel>
                                    <ItemsControl.ItemTemplate>
                                        <DataTemplate>
                                            <Border Background="DodgerBlue"
                                                    Margin="4"
                                                    CornerRadius="4"
                                                    Padding="4 0 4 2">
                                                <TextBlock Text="{Binding Name}"
                                                           Foreground="White" />
                                            </Border>
                                        </DataTemplate>
                                    </ItemsControl.ItemTemplate>
                                </ItemsControl>
                            </StackPanel>
                        </Border>
                    </DataTemplate>
                </syncfusion:SfKanban.CardTemplate>
                <syncfusion:KanbanColumn Categories="Поточні" Title="Поточні" />
                <syncfusion:KanbanColumn Categories="Майбутні" Title="Майбутні" />
                <syncfusion:KanbanColumn Categories="Виконані" Title="Виконані" />

            </syncfusion:SfKanban>
        </Border>
    </DockPanel>
</mah:MetroWindow>
```

ДОДАТОК Б

(обов’язковий)

КОПІЇ НАУКОВИХ ПУБЛІКАЦІЙ

Міністерство освіти і науки України
Хмельницький національний університет

ЗБІРНИК НАУКОВИХ ПРАЦЬ

за матеріалами XIII Всеукраїнської науково-практичної конференції
«Актуальні проблеми комп'ютерних наук АПКН-2021»

15-16 жовтня 2021

Хмельницький 2021

Левчик Т. С., Собко О. В., Житкевич В. В., Міхалевський В. Ц.
Метод автоматизованого діагностування хвороб рослинних культур 359

Манзюк Е. А., Скрипник Т. К.
Система цільової кластеризації на послідових даних 364

Матвійчук І. І., Багрій Р. О., Скрипник Т. К.
Моделювання web-орієнтованих систем 367

Мельник В. С., Міхалевський В. Ц., Скрипник Т. К.
Інформаційна система для комплексної обробки деревини 372

Огнєвий О. В., Медведчук В. Ю., Медведчук Н. К.
Основні принципи організації і особливості відеоконференцзв'язку 375

Онишко О. Г.
Метод та програмні засоби препроцесінгу вхідного текстового контенту 379

Radiuk P. M.
A mental model approach for making decisions in it project management 381

Пасічник О. А.
Програмна система методу вимірювання лінійних переміщень за аналізом зображень 385

Павловський В. І., Савосько О. М.
Виявлення шкідливого трафіку за використанням глибинного навчання 390

Пасічник О. А., Ющенко В. Б., Скрипник Т. К.
Інформаційні технології як засіб автоматизації та оптимізації маркетингових кампаній в соціальних мережах 395

Петровський С. С.
Метод зваженої оцінки успішності навчання у школі 398

Рожков Д. В., Петровський С. С., Скрипник Т. К.
Інформаційна система організації обігу нормативних документів 401

Скрипник Т. К., Манзюк Е. А.
Метод машинного навчання для визначення якості перекладу текстової інформації 404

Ющенко В. Б., Скрипник Т. К, Пасічник О. А.
Інформаційні технології у соц-медіа: PR, реклама, лідогенерація 406

Яковчук М. В., Міхалевський В. Ц., Скрипник Т. К.
Система прийняття рішень у виробничих процесах сільськогосподарського підприємства. 408

Яшина О. М., Мартинюк О. Р.
Система управління якістю у розробці програмних продуктів 410

УДК 004.051

Яшина О. М., Мартинюк О. Р.

Хмельницький національний університет

СИСТЕМА УПРАВЛІННЯ ЯКІСТЮ У РОЗРОБЦІ ПРОГРАМНИХ ПРОДУКТІВ

Розглянуто аспекти проектування системи управління якістю у розробці програмного забезпечення, що включає засоби оцінки та покращення якості програмного коду, а також засоби підвищення якості процесів розробки програмного забезпечення. Запропонована система допомагає інтегрувати принципи загального управління якістю у життєвий цикл програмного забезпечення.

The aspects of the development of software quality control system are overviewed. They include the tools for estimation and improvement of the software code quality and also the tools for improving the software development processes within the company. The proposed informational system helps to integrate the total quality management principles it the software development life cycle.

Високий рівень якості програмного продукту є головним фактором успіху цього продукту і команди її розробників, оскільки даний атрибут безпосередньо впливає на фінансові доходи та видатки, ступінь довіри користувачів, рівень їх задоволеності результатами.

Поняття якості відображає ступінь задоволеності кінцевих користувачів продуктом – тому воно є комплексним, оскільки включає в себе множину різних характеристик, яким програмне забезпечення повинно відповідати. Оскільки в умовах сучасного ринку слід неперервно удосконалювати розроблюваний продукт, існує попит на розробку, покращення та інтеграцію принципів та методів управління якістю програмного забезпечення.

При регламентації розробки програмного забезпечення у командах, менеджери компаній опираються на існуючі методології розробки програмного забезпечення, які не завжди акцентують увагу на процесах управління якістю, що призводить до невідповідності готового продукту його вимогам. Для уникнення даної проблеми, слід інтегрувати принципи, які використовуються у сфері управління якістю при виробництві товарів та розробити інструменти контролю виконання даних принципів.

Метою роботи є проектування системи управління якістю у розробці програмного забезпечення, що інтегрує методи загального управління якістю та

пропонує автоматизовані інструменти контролю якості різних аспектів самого продукту, а також якість організації всіх процесів, що проходять у компанії.

Загальне управління якістю (Total Quality Management – TQM) це один із підходів до управління якістю. TQM включає в себе різні принципи та способи організації процесів управління та контролю якості, основними із яких є акцентування уваги на оптимізації виробничих процесів, неперервне підвищення якості, концентрація на вимогах користувача та інші.

Дана методологія давно себе зарекомендувала як ефективний спосіб підвищення якості у будь-якій сфері. Із точки зору створення розробки програмного забезпечення важливу роль грають інструменти оцінки, підвищення якості продукту та підвищення якості процесів компанії. Дані принципи слід інтегрувати в систему управління якістю у розробці програмного забезпечення, модель якої зображена на рисунку 1.

Рисунок 1 – Система управління якістю у розробці програмних продуктів

Для покращення якості програмного забезпечення використовують автоматизований обрахунок та порівняння різноманітних програмних метрик, розробку та застосування різних типів тестувань, як ручних, так і автоматизованих, також широкого використання набув огляд коду.

Підвищення якості процесів безпосередньо впливає як на якість готового продукту, так і на затрати при розробці, оскільки ефективна організація всіх стадій життєвого циклу програмного забезпечення дозволяє уникати багатьох проблем та

правильно використовувати наявні ресурси. Найбільш широкого розповсюдження набули наступні засоби:

- причинно-наслідкова діаграма;
- контрольна карта;
- діаграма Парето;
- діаграма спорідненості;
- матриця пріоритетів.

Отже, запропонована система управління якістю у розробці програмних продуктів забезпечує засоби визначення оцінки якості продукту та організаційних процесів, а також методи підвищення якості усіх процесів, що відбуваються у компанії. Подальші дослідження спрямовані на автоматизацію процесу аналізу програмного коду та інтеграцію системи у модель життєвого циклу програмного забезпечення.

Перелік посилань

1. Daniel Galin. Software quality. Concepts and practice / Wiley-IEEE Press, 2018. – 697 p.
2. Poornima M. Charantimath. Total Quality Management/ Pearson Education, 2017. – 640 p.
3. The impact of Total Quality Management on organizational performance [Електронний ресурс]. – Режим доступу: https://www.researchgate.net/publication/294886200_The_impact_of_Total_Quality_Management_on_organizational_performance

УДК 004.051
DOI:

МАРТИНЮК О. Р.
Хмельницький національний університет
ORCID ID: 0000-0003-0383-1750
e-mail: olexandr.martynyuk1@gmail.com
ЯШИНА О. М.
Хмельницький національний університет
ORCID ID: 0000-0001-7816-1662
e-mail: oksana.yashyna@ukr.net
РАДЕЛЬЧУК Г. І.
Хмельницький національний університет
ORCID ID: 0000-0002-9728-4390
e-mail: gal_2015@ukr.net
КУСТОВСЬКИЙ Р.С.
Хмельницький національний університет
ORCID ID: 0000-0003-0785-7202
e-mail: roma.kust99@gmail.com

ПОРІВНЯННЯ ПРОГРАМНИХ МЕТРИК ДЛЯ ОЦІНКИ ЯКОСТІ ПРОГРАМНИХ ПРОДУКТІВ

У статті наведено результати досліджень різних типів метрик програмного забезпечення та конкретних представників даних типів. Описано сферу їх застосування та зручність використання як при оцінці якості готового продукту, так і на стадії розробки.

Ключові слова: управління якістю, контроль якості, програмне забезпечення, метрика програмного забезпечення, програмний продукт.

OLEKSANDR RUSLANOVYCH MARTYNIUK, OKSANA MYKOLAIVNA YASHYNA., GALYNA IVANIVNA RADELCHUK, ROMAN SERGIYOVYCH KUSTOVSKYI
Khmelnytskyi National University

COMPARISON OF SOFTWARE METRICS FOR QUALITY ESTIMATION OF THE SOFTWARE PRODUCTS

There is a well known principle, that you can manage only what you can measure – therefore adequate measurement and estimation of different elements in any engineering area is a very important development factor, and software engineering is not an exception.Software metrics represent numeric values of software characteristics and are used for analysis of different aspects of the product in order to predict possible problems. Software metrics can be applied to any stage of the software life cycle. For instance, on a stage of requirement analysis and the problem formulation, these metrics help to estimate the required amount of work and, respectively, the amount of financial and work resources. In addition, the metrics can be used directly in estimation of the program code complexity to assess it's performance, effectiveness and the ability to modify it if needed. Software metrics are an important part of the whole software quality management process as they can be easily automated and applied in order to achieve some valuable technical characteristics. Moreover, they can be combined with another quality management tools and are a beneficial addition to any software product development model.In general, measuring the attributes of software during any phase of development gives a number of beneficial results to the developers as it helps to use the time, finances and effort effectively. This is achieved because of early problems identification, estimation and prevention and because of that the

size, cost, quality and simplicity of the maintenance of the product is always under control. This article describes the results of research on different software metrics types and the actual representatives of these types. It assesses the usage area and the convenience of it's usage on evaluating the quality of both a finished product and a product in the development stage.

Keywords: quality management, quality control, software, software metric, software product

Вступ. Постановка проблеми

Адекватне вимірювання та оцінка різних елементів у будь-якій галузі інженерії є важливим фактором розробки. Інженерія програмного забезпечення не є виключенням, оскільки загальновідомим є принцип – керувати можна лише тим, що можна виміряти, тому

Метрики програмного забезпечення (ПЗ) представляють собою чисельні значення характеристик ПЗ та використовуються для аналізу різних аспектів продукту з метою передбачення можливих проблем. Програмні метрики можуть бути використані на будь-якому етапі життєвого циклу ПЗ. Наприклад, на етапі аналізу вимог та постановки задачі, метрики допомагають оцінити необхідний обсяг робіт та, відповідно, фінансові і трудові ресурси. Також метрики можуть використовуватись безпосередньо для оцінки складності програмного коду, для аналізу його швидкодії, ефективності та можливості адаптації у разі необхідності. Програмні метрики є важливою частиною всього процесу управління якістю ПЗ, оскільки вони можуть бути легко автоматизовані та застосовані з метою досягнення важливих технічних характеристик. Більше того, вони можуть бути поєднані з іншими засобами управління якістю і є корисним доповненням до моделі розробки будь-якого ПЗ.

Загалом, вимірювання характеристик ПЗ на будь-якій стадії розробки приносить низку корисних результатів для розробників, оскільки допомагає ефективно використовувати час, фінанси та зусилля. Це досягається шляхом раннього розпізнання, оцінки та запобігання проблемам; за рахунок цього розмір, вартість, якість та простота підтримки продукту завжди буде під контролем.

Аналіз останніх досліджень та публікацій

Основою дослідження є праці науковців у сфері управління якістю ПЗ, управління проектами ПЗ та міжнародні стандарти серії ISO/IEC щодо оцінки і вимірювання якості програмних продуктів. Згідно з дослідженнями, які описані у працях [1, 2], оцінка та забезпечення якості ПЗ є широкою темою, тому існує багато різних підходів до оцінки якості програмних продуктів. Загалом, поняття якості трактується як ступінь відповідності кінцевого продукту вимогам замовника. У галузі ПЗ визначені наступні основні критерії якості [3]:

- правильність;
- ефективність;
- цілісність;
- зручність використання;
- зручність підтримки;
- гнучкість;
- зручність тестування;
- можливість повторного використання.

У процесі розробки програмного продукту враховуються усі важливі критерії та їх пріоритети, у результаті чого підбирається модель управління якістю ПЗ, яка передбачає використання конкретних принципів розробки, метрик та методів тестування.

Метою роботи є: порівняння та аналіз метрик програмного забезпечення і типів, до яких вони відносяться, на предмет сфери застосування та зручності використання.

Виклад основного матеріалу

Метрики програмного забезпечення – це стандартизовані вимірювання певних характеристик програмної системи чи процесу з метою визначення їх ступеня та міри впливу на продукт у цілому. Метою обрахунку програмних метрик є отримання об'єктивних та зрозумілих вимірювань для подальшого

використання при плануванні бюджету, плану розробки, тестуванні, оптимізації та документації ПЗ [4]. Застосування метрик надає низку переваг для розробників, оскільки це спрощує прийняття вагомих рішень щодо архітектури програмного продукту, використання тих чи інших фреймворків, мов програмування тощо. Також це дозволяє швидко розпізнавати та усувати можливі несправності продукту, дає чітке і повне розуміння швидкодії програми та шляхи її оптимізації.

Тим не менше, оскільки процес проектування та розробки програмного продукту є доволі складним і комплексним, метрики ПЗ виступають лише інструментом при аналізі проекту спеціалістом та мають певні обмеження. Наприклад, застосування метрик не завжди є простим процесом, в багатьох випадках це складно та дорого; окрім того, кожен проект є унікальним. Саме тому загальновідомі метрики можуть бути використані лише для вимірювання базових характеристик програмного коду, а для глибшого аналізу слід розробити метрику конкретно для даного програмного продукту.

Метрики програмного забезпечення є емпіричними, а тому не покривають весь спектр атрибутів коду і відображають не абсолютну оцінку ПЗ, а лише ймовірність виникнення проблем та збоїв [5].

Методологія даного дослідження полягає у використанні наступних критеріїв порівняння метрик ПЗ:

- основна ідея метрики та галузь її застосування;
- простота реалізації та гнучкість використання;
- основні переваги та недоліки метрик ПЗ.

Класифікація розглянутих у даній статті програмних метрик представлена на рисунку 1.

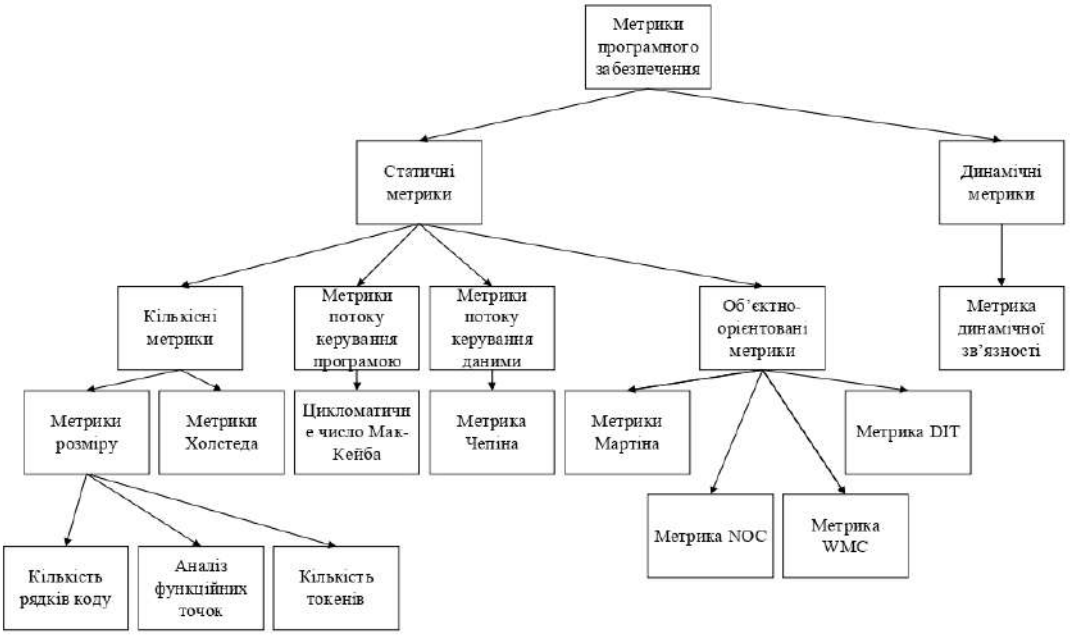


Рис. 1. Класифікація метрик програмного забезпечення

У сучасних комплексних проектах знаходять своє застосування представники всіх типів програмних метрик, однак кожна з них має свої особливості застосування та може трактуватись по-різному, у залежності від специфіки конкретного програмного продукту.

Метрика динамічної зв'язності (Dynamic Coupling Metric – DCM) використовується для вимірювання зв'язності між парою об'єктів чи класів під час роботи програмної системи [6]. Перевагою цієї метрики є інформативність, оскільки її можна використовувати на рівні модулів та виміряти ступінь залежності між цими модулями. До недоліків метрики відноситься складність використання та неможливість її застосування на

ранніх етапах розробки ПЗ, оскільки у наявності має бути робоча система з відомою архітектурою.

Кількість рядків коду (Source Lines of Code – SLoC) – це метрика розміру програмного продукту, основна ідея якої полягає у підрахунку кількості рядків коду, без врахування коментарів та порожніх рядків [7]. Ця метрика розроблена для оцінки зусиль, витрачених на розробку програмного модуля та для порівняння продуктивності розробників. Метрика SLoC є однією з найпростіших і ранніх метрик, тому вона має низку обмежень. Основними з цих обмежень є те, що одна й та сама функціональність може бути як записана в один рядок, так і розбита на декілька; також потрібно враховувати специфіку окремих мов програмування. Враховуючи це, дану метрику не можна вважати гнучкою, проте вона є простою у реалізації та має широку галузь застосування. Розрізняють два типи рядків – фізичні та логічні; фізичні рядки представляють кількість усіх рядків коду, а логічні – кількість команд програми.

Кількість токенів (Token Count) – концепція цієї метрики полягає у тому, щоб розглядати програму як колекцію токенів. Токенами вважаються унікальні оператори та операнди, і вся логіка роботи програми може бути визначена за допомогою цих базових конструкцій [1]. Розмір програми вважається сумою кількості унікальних операторів та кількості унікальних операндів. Метрика Token Count має широку галузь застосування, оскільки вона може бути інтегрована у будь-який проект. Проте ця метрика не враховує специфіки архітектури конкретної програмної системи, а тому вона може використовуватись лише як додатковий інструмент, а не як самостійний критерій оцінки ПЗ.

Аналіз функціональних точок (Function Point) – виражає обсяг бізнес-логіки ПЗ за допомогою аналізу функцій та модулів системи. Усі функції програми розподіляються на п'ять типів: функції введення, функції виведення, запити, функції роботи з файлами та зовнішні інтерфейси. Після розподілу на категорії проводиться аналіз алгоритмічної складності кожної функції, на основі чого формується розмір програмного продукту [2]. Перевагою цієї метрики є її універсальність – результати оцінки не залежать від мови програмування та архітектури додатку; проте ця метрика є доволі складною у використанні.

Метрики Холстеда (Halstead Metrics) також відносяться до кількісних метрик, проте вони відображають значно більше інформації у порівнянні з попередніми метриками. Цей метод передбачає підрахунок загальної кількості операторів, загальної кількості операндів, кількості унікальних операторів та кількості унікальних операндів; за допомогою цих показників розраховуються значення розміру програми, розміру словника, складність програми, обсяг зусиль тощо [1]. До переваг Halstead Metrics можна віднести відносну простоту використання та універсальність, оскільки ці метрики не залежать ні від мови програмування, ні від складності алгоритму, що описується. До недоліків метрик Холстеда можна віднести те, що вони є статичними, тому вони обраховуються безпосередньо за допомогою коду, і, отже, не відображають атрибути програмної системи під час її виконання.

Найпоширенішим представником ***метрик потоку керування програмою*** є цикломатична складність програми, або ***цикломатичне число Мак-Кейба*** (McCabe's Cyclomatic Metric). Концепція цієї метрики полягає у представленні програми як зв'язного орієнтованого графа, вузли якого відображають частини вихідного коду, а ребра – потік керування, який виконується під час роботи програми. Цикломатичне число рівне числу лінійно незалежних шляхів роботи програми у його відображенні за допомогою графа. Ця метрика описує складність роботи програми та число можливих відгалужень алгоритму роботи під час її виконання.

Метрика Чепіна (Chapin's Metric) – метрика ***потоку керування даними***, суть якої полягає в оцінці інформаційної єдності окремо взятого програмного модуля за допомогою аналізу використання змінних введення та виведення. При цьому змінні поділяються на чотири типи: вхідні змінні, створені змінні, керуючі змінні та паразитні змінні. Ця метрика є доволі корисною при аналізі програмних модулів на оптимальність роботи з даними, проте її складно обчислити, а поділ змінних потребує від розробника додаткового аналізу.

Широкого використання набувають ***об'єктно-орієнтовані метрики***, оскільки об'єктно-орієнтована парадигма програмування на сьогодні є досить поширеною. Серед цих метрик можна виділити наступні. ***Метрики Мартіна*** (Martin's Metrics) допомагають оцінити рівень нестабільності класу, рівень абстракції та залежність нестабільності класу від його абстракції. ***Метрика WMC*** (Weighted Methods per Class) представляє

собою сумарну складність усіх методів класу, ***метрика DIT*** (Depth of Inheritance Tree) – глибину дерева наслідування, ***метрика NOC*** (Number of children) – кількість класів, що наслідують даний [8].

Об'єктно-орієнтовані метрики дають змогу оцінити оптимальність описаних класів та інтерфейсів з метою забезпечення хорошого рівня абстракції та зв'язності між об'єктами класів, однак ці метрики є вузькоспеціалізованими і можуть бути використані лише при аналізі ПЗ у межах об'єктно-орієнтованої парадигми програмування.

В цілому очевидно, що для управління якістю конкретного програмного продукту не можна обмежуватись використанням однієї метрики чи одного типу метрик, оскільки вони покликані вирішувати різні завдання проектування ПЗ. Частина програмних метрик є універсальними, а інші використовуються лише при розробці програм у певній парадигмі програмування. Тому оптимальним підходом є комбінування різних програмних метрик та розробка власних при потребі глибшої та якіснішої оцінки якості ПЗ.

Висновки

У дослідженні виконано порівняння та аналіз різних типів метрик ПЗ та конкретних представників цих типів на предмет галузі застосування, концепції окремих метрик, а також зручності та гнучкості використання. Визначені основні переваги та недоліки метрик ПЗ при використанні у проектах та особливості їх реалізації.

За результатами дослідження встановлено, що жодна метрика не може вирішити завдання виміру якості програмного продукту у повній мірі, а тому оптимальним є використання різних типів метрик для отримання детальної оцінки ПЗ. Підхід, який полягає у поєднанні різних програмних метрик у залежності від парадигми, архітектури, а також етапу життєвого циклу ПЗ, дозволяє досягати високої якості програмного продукту.

Література

1. A Study of Software Metrics [Electronic resource] // ResearchGate. – URL: https://www.researchgate.net/-publication/322070697_A_Study_of_Software_Metrics

2. Analysis of Software Quality Using Software Metrics [Electronic resource]. – Mode of acces: https://www.researchgate.net/publication/328830202_Analysis_of_Software_Quality_Using_Software_Metrics

3. Software Quality Factors and Software Quality Metrics [Electronic resource]. – Mode of acces: https://www.researchgate.net/publication/263582173_Software_Quality_Factors_and_Software_Quality_Metrics_to_Enhance_Software_Quality_Assurance

4. Galin D. Software quality. Concepts and practice / D. Galin. – Publisher: Wiley-IEEE Press, 2018. – 720 p.

5. Garst Smith Howard T. Software quality assurance: A guide for developers and auditors / Howard T. Garst Smith. – Publisher: CRC Press Inc, 2020. – 480 p.

6. Suryn W. Software quality engineering. A practitioner's approach / Suryn W. – Publisher: Wiley-IEEE Computer Society Pr, 2014. – 208 p.

7. ISO/IEC 25023:2016 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality [Electronic resource] // ISO.org. – Mode of acces: https://www.iso.org/ru/standard/35747.html

8. Рыжков Е. Программный код и его метрики [Электронный ресурс] / Е. Рыжков // Веб-портал Хабр. – Режим доступа: https://habr.com/ru/company/intel/blog/106082/

References

1. A Study of Software Metrics [Electronic resource] // ResearchGate. – URL: https://www.researchgate.net/-publication/322070697_A_Study_of_Software_Metrics

2. Analysis of Software Quality Using Software Metrics [Electronic resource]. – Mode of acces: https://www.researchgate.net/publication/328830202_Analysis_of_Software_Quality_Using_Software_Metrics

3. Software Quality Factors and Software Quality Metrics [Electronic resource]. – Mode of acces: https://www.researchgate.net/publication/263582173_Software_Quality_Factors_and_Software_Quality_Metrics_to_Enhance_Software_Quality_Assurance

4. Galin D. Software quality. Concepts and practice / D. Galin. – Publisher: Wiley-IEEE Press, 2018. – 720 p.

5. Garst Smith Howard T. Software quality assurance: A guide for developers and auditors / Howard T. Garst Smith. – Publisher: CRC Press Inc, 2020. – 480 p.

6. Suryn W. Software quality engineering. A practitioner's approach / Suryn W. – Publisher: Wiley-IEEE Computer Society Pr, 2014. – 208 p.

7. ISO/IEC 25023:2016 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Measurement of system and software product quality [Electronic resource] // ISO.org. – Mode of acces: https://www.iso.org/ru/standard/35747.html

8. Ryzhkov E. Programmnyj kod i ego metriki [Elektronnyj resurs] / E. Ryzhkov // Veb-portal Habr. – Rezhim dostupa: https://habr.com/ru/company/intel/blog/106082/

ДОДАТОК В

(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Удосконалення методу управління якістю розробки програмних продуктів

Автор роботи:
ст. гр. ІПЗм 20-1 Мартинюк О. Р.
Керівник роботи:
д. т. н., проф Бармак О. В.

Об'єктом дослідження є процеси життєвого циклу програмного забезпечення.

Предметом дослідження є методи та підходи управління якістю розробки програмних продуктів.

Метою дослідження є удосконалення методу управління якістю розробки програмних продуктів.

Актуальність теми полягає у тому, що на даний момент відбуваються активні пошуки кращих стратегій ефективної розробки програмних продуктів та забезпечення їх якості.

Дане дослідження концентрується на вирішенні проблеми відсутності контролю ефективності процесів розробки сучасними підходами у розробці програмного забезпечення, а рішення базується на вирішенні даної проблеми шляхом впровадження системи менеджменту якості розробки програмних продуктів.

Завдання дослідження

- провести аналіз моделей розробки програмного забезпечення із точки зору процесів управління якістю;
- проаналізувати системи менеджменту якості, вітчизняні та міжнародні стандарти;
- покращити метод управління якістю розробки програмних продуктів шляхом впровадження системи менеджменту якості;
- сформувати компоненти системи управління якістю розробки програмного забезпечення згідно серії стандартів ДСТУ ISO 9000 та принципів TQM;

Завдання дослідження

- провести аналіз існуючих методів контролю якості програмного коду на предмет оптимізації;
- провести аналіз існуючих методів контролю якості організаційних процесів на предмет оптимізації;
- виконати проєктування програмної реалізації розробленої системи;
- розробити програмну реалізацію;
- провести тестування та аналіз запропонованого методу управління якістю розробки програмних продуктів.

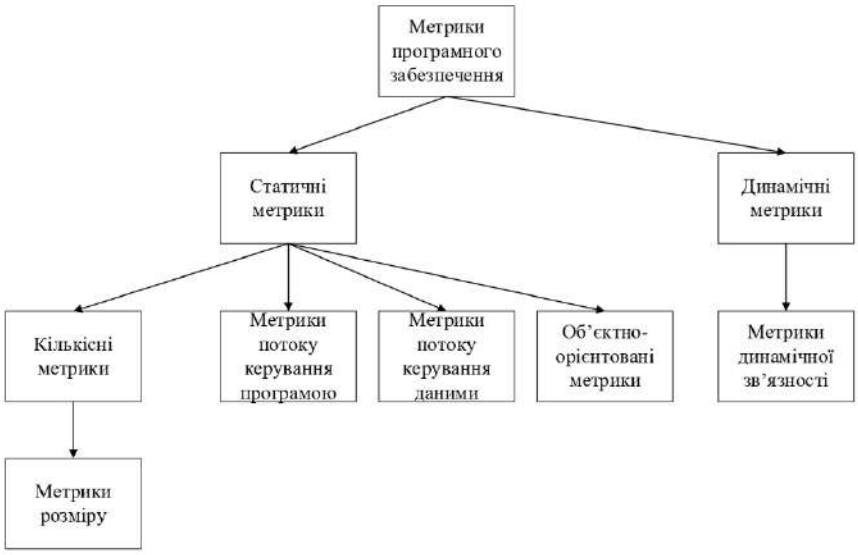
Аналіз стану проблеми та існуючих рішень

Емпіричні дослідження, розглянуті в ході аналізу предметної області, приводять до висновків про високу ефективність застосування принципів TQM та ДСТУ ISO 9000 у всіх сферах промисловості, тому дані принципи варто інтегрувати і в процеси управління якістю, що практикуються сьогодні при розробці програмного забезпечення. Актуальні на даний момент підходи не звертають увагу на покращення організаційних процесів всередині компаній, що веде до неефективного використання ресурсів та зниження якості кінцевого програмного продукту, проте за допомогою впровадження системи менеджменту якості на основі TQM та відповідно до серії стандартів ДСТУ ISO 900, можна вирішити дану проблему.

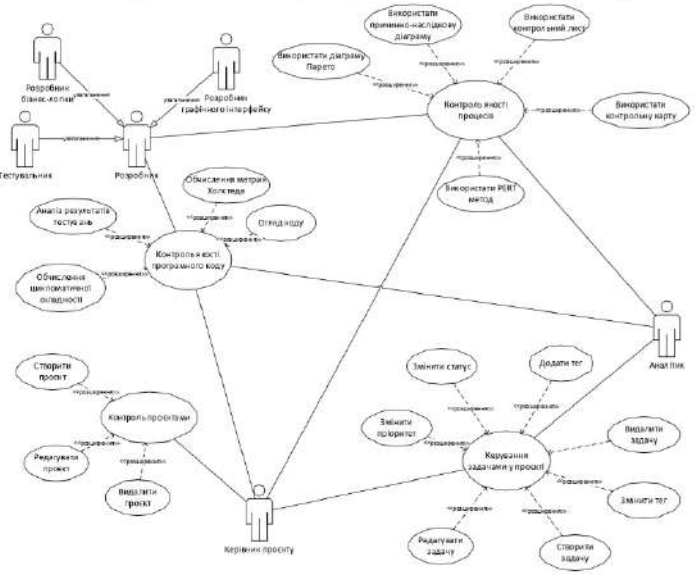
Принципи управління якістю згідно ДСТУ ISO 9000

Модель системи управління якістю розробки програмних продуктів

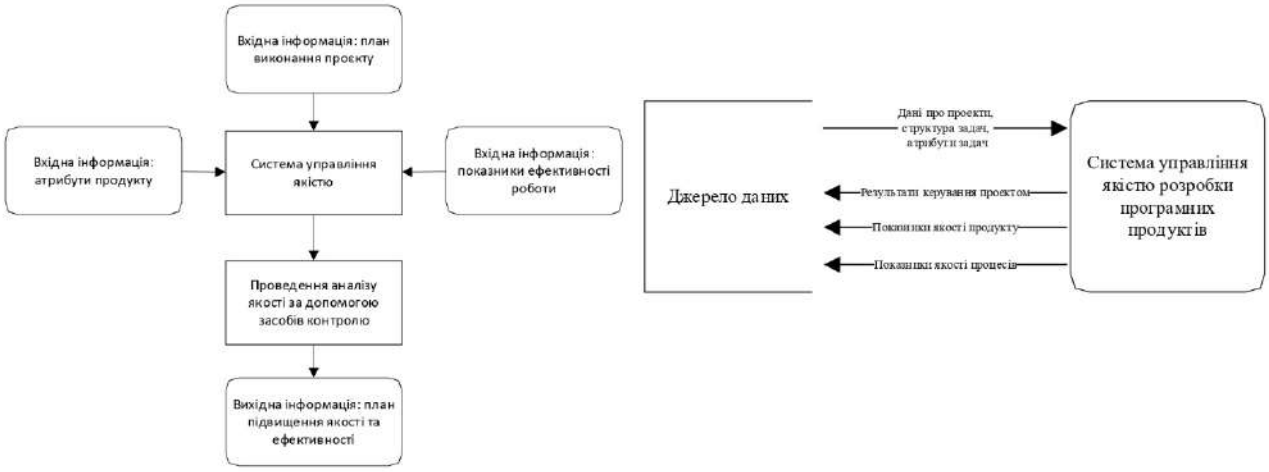
Класифікація метрик програмного забезпечення



Діаграма варіантів використання розроблюваної програмної реалізації

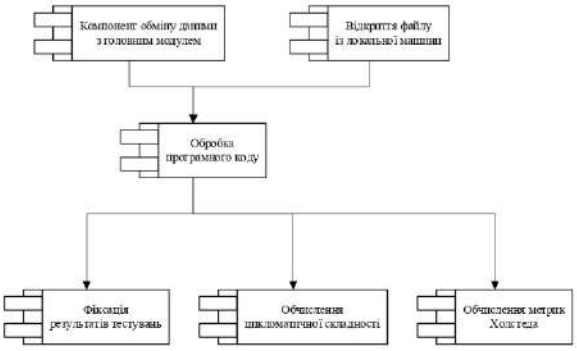


Структура системи та діаграма потоків даних

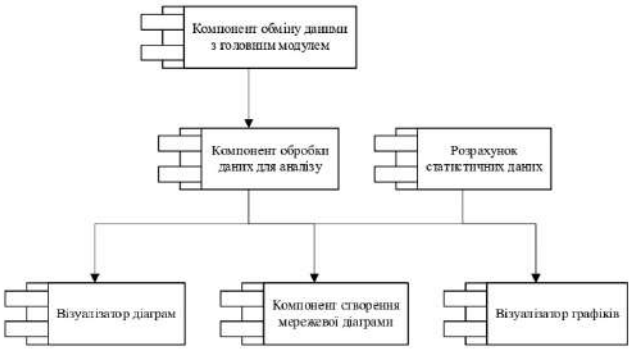


Діаграми компонентів модулів системи

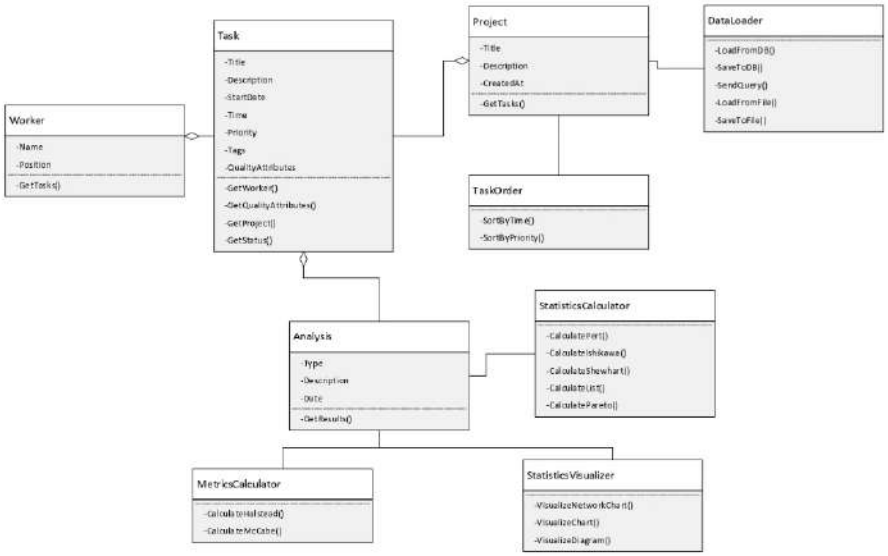
Модуль контролю якості коду



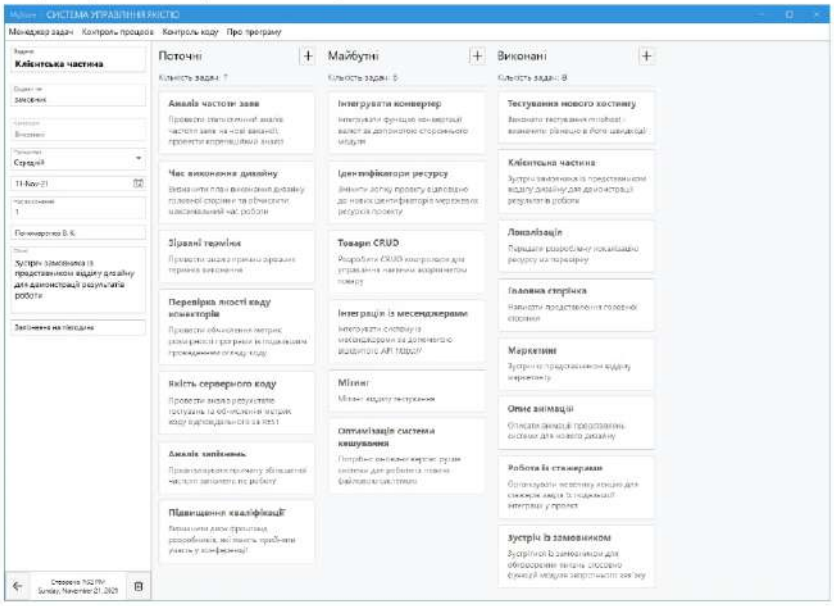
Модуль контролю якості процесів



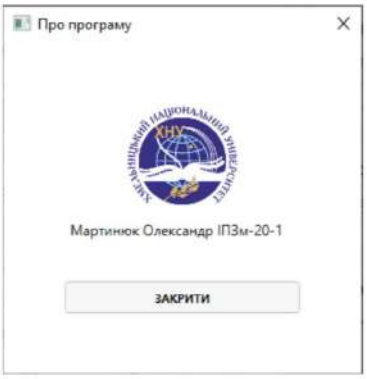
Діаграма класів інформаційної системи



Програмна реалізація системи

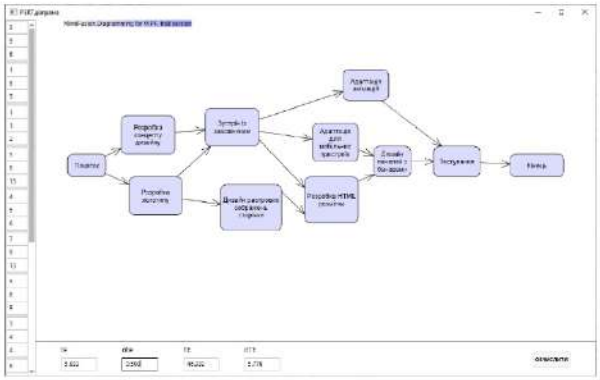


Панель управління задачами

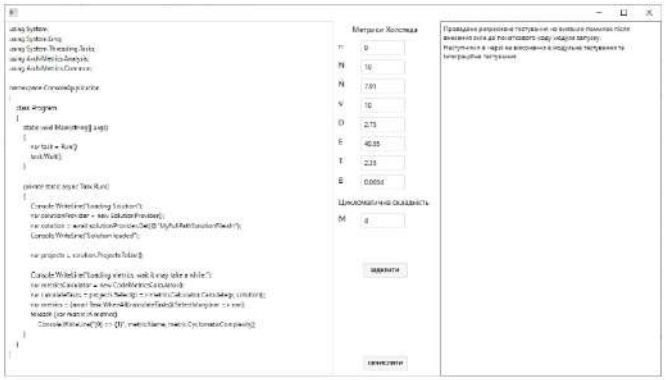


Про програму

Програмна реалізація системи



PERT діаграма



Засоби контролю якості коду

Наукова новизна отриманих результатів:

- отримав подальший розвиток підхід загального управління якістю у напрямку його застосування у сфері розробки програмних продуктів, що дозволяє підвищити якість програмного коду та процесів його розробки;
- удосконалено систему менеджменту якості TQM у сфері розробки програмних продуктів, що поєднує як засоби аналізу якості коду, так і методи статистичного аналізу якості процесів;
- удосконалено метод управління якості розробки програмних продуктів.

Практичне значення отриманих результатів полягає у вдосконаленні методу управління якістю розробки програмних продуктів, шляхом впровадження отриманої системи менеджменту якості у процес розробки, що дає змогу суттєво покращити ефективність роботи команди шляхом підвищення якості коду та оптимізації організаційних процесів.

Доцільність та ефективність розробки системи теоретично обґрунтована у першому та другому розділах кваліфікаційної роботи опираючись на результати емпіричних досліджень.

Висновки:

- проведено порівняльний аналіз моделей розробки програмного забезпечення та існуючих систем менеджменту якості;
- сформовано компоненти системи управління якістю розробки програмного забезпечення згідно серії стандартів ДСТУ ISO 9000 та принципів TQM;
- сформовано набір засобів контролю якості програмного коду та засобів контролю якості організаційних процесів;
- спроектовано та реалізовано програмну імплементацію;
- протестовано та проаналізовано запропоновану систему;
- удосконалено метод управління якістю розробки програмних продуктів.

За темою кваліфікаційної роботи опублікована наукова стаття: Порівняння метрик оцінки якості програмних продуктів / Мартинюк О. Р., Яшина О.М., Радельчук Г.І., Кустовський Р.С. // Вісник Хмельницького національного університету. Технічні науки. – Хмельницький, 2021. – №5.

За темою роботи опубліковані тези: Система управління якістю у розробці програмних продуктів / Мартинюк О.Р., Яшина О.М. // Збірник наукових праць за матеріалами XIII всеукраїнської науково-практичної конференції «Актуальні проблеми комп'ютерних наук АПКН-2021». – Хмельницький, 2021.

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Мартинюка О. Р.
Прізвище, ініціали

факультет ІТ, 2 курс, група ІПЗм-20-1

ЗАЯВА

З правилами чинного Положення «Про дотримання академічної доброчесності в Хмельницькому національному університеті» від 26.09.2020 (зі змінами від 26.11.2020), згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений (а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

23.11.2021 р.
дата

підпис

Thu Dec 02 14:31:10 EET 2021, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 21.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 7%**

	Документ		Сумарний збіг по Базі Даних	
ID: 97847 Назва: Удосконалення методу управління якості розробки програмних продуктів Додано в БД: 2021-12-02 Автора: О. Р. Мартинюк Керівники: О. В. Бармак Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	115234	777	28237 (25%)	211 (27%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
95867	Назва: Звыт з переддипломноъ практики Додано в БД: 2021-09-27 Автора: О. Р. Мартинюк Керівники: Бармак О.В. Консультанти: Опоненти:	24759 (21.0%)	171 (22.0%)

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1009480565

Дата перевірки:
02.12.2021 15:23:23 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
02.12.2021 15:28:03 EET

ID користувача:
100005589

Назва документа: **Мартинюк_маг_без_додатків**

Кількість сторінок: **86** Кількість слів: **15617** Кількість символів: **127583** Розмір файлу: **1.13 MB** ID файлу: **1009494330**

4.68%
Схожість

Найбільша схожість: **2.5%** з джерелом з Бібліотеки (ID файлу: **1009494327**)

2.43% Джерела з Інтернету	324	Сторінка 88
3.51% Джерела з Бібліотеки	88	Сторінка 90

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	6

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА ДИПЛОМНУ РОБОТУ

Дипломник студент групи ІПЗм-20-1 Мартинюк О. Р.

Тема Удосконалення методу управління якістю розробки програмних продуктів

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг дипломної роботи:

Кількість листів креслень 0; кількість сторінок записки 82

1.Короткий зміст ДР та прийнятих рішень Представлена робота присвячена актуальній темі в розробці програмного забезпечення і складається з наступних розділів: вступ, аналіз сучасних методів управління якості розробки програмних продуктів, модель та методи системи управління якості розробки програмних продуктів, архітектура програмної реалізації системи управління якості розробки програмних продуктів, програмна реалізація системи управління якості розробки програмних продуктів, висновки, додатки.

2. Висновок про відповідність ДР поставленому завданню Магістерська кваліфікаційна робота виконана у відповідності до завдань із дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі студент провів детальний аналіз предметної області, дослідив етапи розробки програмного забезпечення, провів порівняння існуючих моделей із системами менеджменту якості, на основі яких довів актуальність роботи. У другому розділі було сформовано компоненти системи управління якістю розробки програмних продуктів, обґрунтовано, чому використання розроблюваної системи найкраще підходить для вирішення проблеми. В третьому розділі автором було виконано детальне проектування системи із використанням багатьох діаграм. У четвертому розділі здійснено вибір інструментів розробки, опис програмної реалізації і тестування розробленого продукту на основі чого були підведені підсумки щодо практичної значимості проведеної роботи.

4. Позитивні сторони роботи До позитивних сторін роботи слід віднести глибоке дослідження усіх існуючих підходів у сфері управління якості як програмного забезпечення, так і за його межами. Чітке обґрунтування усіх прийнятих в ході виконання роботи рішень, а також приділення особливої уваги дослідженню емпіричних наукових досліджень, проведених за тематикою кваліфікаційної роботи.

5. Негативні сторони роботи В ході рецензування виявлено недоліки по оформленню представленого матеріалу та незначні проблеми у описі деяких частин роботи, які були усунені.

6. Оцінка графічного оформлення та пояснювальної записки роботи Представлені матеріали роботи є правильно виконаними та логічно структурованими, що відображає послідовність виконання поставлених завдань. Проте була присутня певна кількість

стилістичних помилок, які згодом були виправлені, Таким чином виконання пояснювальної записки та графічного оформлення заслуговує оцінки «добре».

7. Відгук про роботу в цілому Зміст представленої роботи не лише в повній мірі розкриває обрану тему, а і пропонує логічно обгрунтовану стратегію щодо покращення існуючого методу. Проведені дослідження в достатній мірі аргументовані, виконані на високому теоретичному та практичному рівні. Результатом проведення досліджень стали відповідні висновки і конкретні пропозиції щодо вдосконалення методу управління якістю розробки програмних продуктів.

8. Інші зауваження

9. Оцінка дипломної роботи Робота заслуговує оцінки «добре», а її автор – присвоєння кваліфікації «магістра» з інженерії програмного забезпечення.

РЕЦЕНЗЕНТ (прізвище, ім'я, по-батькові, посада, місце роботи) Говорущенко Тетяна Олександрівна, доктор технічних наук, професор, зав. кафедри комп'ютерної інженерії та системного програмування ХНУ.

"29" листопада 2021 р. _____
(підпис)

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: «Удосконалення методу управління якістю розробки програмних продуктів»

Автор: Мартинюк Олександр Русланович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Бармак Олександр Володимирович, д. т. н., професор
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Сумарний обсяг всіх запозичень, визначений системами виявлення збігів ідентичності схожості, а саме:

- системою Anti-Plagiarism, складає 21.0% і адресується до 1 джерела, а саме 2-го розділу звіту з переддипломної практики студента, який є складовою частиною роботи;
- за програмою UNICHECK виявлені 4.68%.; найбільша схожість: 2.5% з джерелом з Бібліотеки (ID файлу: 1009494327); інші схожості є фрагментарними – містять поширені конструкції, загальновідомі терміни, скорочення та визначення.

Таким чином, всі виявлені запозичення, з урахуванням наведених обґрунтувань, не є плагіатом, Робота приймається до захисту.

Керівник _____ О. В. Бармак

Гарант ОП _____ О. М. Яшина

Завідувач кафедри _____ Л. П. Бедратюк