НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

Факультет прикладної математики

Кафедра прикладної математики

«На правах рукопису»
УДК 519.2 : 004.054

«До захисту допущено»

Завідувач кафедри
_____ О. Р. Чертов

«___» _____2015 р.

Магістерська дисертація

на здобуття ступеня магістра

зі спеціальності 8.04030101 «Прикладна математика»

на тему: Модель оцінки якості програмного забезпечення

Виконав: студент 2 курсу, групи КМ-31М

Довгаль Костянтин Ігорович _____

Науковий керівник	завідувач кафедри, д-р техн. наук, доцент Чертов О. Р.	_____
Консультант із нормоконтролю	старший викладач Мальчиков В. В.	_____
Рецензент	декан ФПМ, професор, д-р техн. наук, проф. Дичка І. А.	_____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2015 року

Національний технічний університет України

«Київський політехнічний інститут»

Факультет прикладної математики

Кафедра прикладної математики

Рівень вищої освіти – другий (магістерський)

Спеціальність 8.04030101 «Прикладна математика»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ О.Р. Чертов

«__» _____ 2015 р.

ЗАВДАННЯ

на магістерську дисертацію студенту

Довгалю Костянтину Ігоровичу

1. Тема дисертації «Модель оцінки якості програмного забезпечення», науковий керівник дисертації Чертов Олег Романович, д-р техн. наук, доцент, затверджені наказом по університету від «20» березня 2015 р. № 785-С.
2. Термін подання студентом дисертації: «18» червня 2015 р.
3. Об'єкт дослідження: процеси формалізації та математичного оброблення текстових даних, які представляють вихідний код програмного забезпечення.
4. Предмет дослідження: математична модель, яка характеризує якість вихідного коду програмного забезпечення.
5. Перелік завдань, які потрібно розробити:

– дослідити способи розробки моделі оцінки якості програмного забезпечення;

– провести аналіз існуючих метрик вихідного коду програмного забезпечення;

– обґрунтувати вибір метрик вихідного коду ПЗ для побудови математичної моделі;

– запропонувати методику комп'ютерного моделювання якості ПЗ;

– розробити та дослідити математичну модель оцінки якості вихідного коду програмного забезпечення.

6. Орієнтовний перелік ілюстративного матеріалу:

– теоретичні аспекти побудови та оптимізації математичної моделі оцінювання якості вихідного коду ПЗ;

– порівняльний аналіз способів побудови відповідної моделі;

– блок-схема алгоритму оцінки якості ПЗ;

– порівняльний графік отриманих результатів;

– екранні форми розроблених програмних засобів;

– часові характеристики роботи програмного модуля.

7. Орієнтовний перелік публікацій:

– VII наукова конференція магістрантів та аспірантів «Прикладна математика та комп'ютинг – ПМК'2015»;

– XVII Міжнародна науково-технічна конференція SAIT 2015;

8. Дата видачі завдання «1» жовтня 2013 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Вибір напряму дослідження та узгодження тематики МД з керівником	15 вересня – 30 жовтня 2013 р.	
2	Ґрунтовне ознайомлення з предметною областю	30 жовтня 2013 – 15 лютого 2014 р.	
3	Вивчення літератури, пошук додаткової інформації	15 лютого – 1 вересня 2014 р.	
4	Проведення дослідження, розроблення програмного забезпечення	1 вересня 2014 – 1 березня 2015 р.	
5	Завершення роботи над основною частиною МД, переддипломна практика, робота над публікаціями	1 березня – 1 травня 2015 р.	
6	Оформлення текстової і графічної частини МД	1 травня – 1 червня 2015 р.	
7	Попередній захист МД	1 червня – 15 червня 2015 р.	

Студент _____ К. І. Довгаль

Науковий керівник дисертації _____ О. Р. Чертов

РЕФЕРАТ

Актуальність теми. Незважаючи на стрімкий та динамічний розвиток IT-індустрії та її успіхи по всьому світу, дисципліна розробки ПЗ все ще не сформувалася як зріла технічна дисципліна. Оскільки етап планування та оцінювання необхідного бюджету та термінів є одним з найбільш принципових у процесі розробки ПЗ, то, враховуючи складність та затратність цього процесу, збільшення точності оцінок при прийнятті управлінських рішень, є доцільним.

Об'єктом дослідження є процеси формалізації та математичного оброблення текстових даних, які представляють вихідний код програмного забезпечення.

Предметом дослідження є математична модель, яка характеризує якість вихідного коду програмного забезпечення.

Мета роботи: розробка і дослідження математичної моделі оцінки якості програмного забезпечення та спеціальних методів математичного моделювання складних системних об'єктів вихідного коду ПЗ для виявлення інтегральних властивостей, що характеризують систему як єдине ціле.

Методи дослідження. В роботі використовуються методи математичного моделювання, методи системного аналізу, чисельні методи.

Наукова новизна полягає в наступному:

- розроблено нову математичну модель оцінки якості ПЗ, яка відрізняється від існуючих врахуванням великої кількості метрик вихідного коду, можливістю наповнення експертними знаннями, що підлягають моделюванню, і у такий спосіб дозволяє виконати більш точну оцінку якості ПЗ.

Практична цінність отриманих у роботі результатів полягає у тому, що запропонована модель та розроблені засоби дають змогу отримати картину

взаємозв'язків і взаємозумовленостей процесів та фактів, які впливають на якість вихідного коду ПЗ. Встановивши причинно-наслідкові зв'язки, можна дати ґрунтовну оцінку якості ПЗ.

Апробація роботи. Основні положення і результати роботи були представлені та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2015 (Київ, 15-17 квітня 2015 р.) та опубліковані у збірнику праць XVII Міжнародної науково-технічної конференції «SAIT 2015».

Структура та обсяг роботи. Магістерська дисертація складається зі вступу, чотирьох розділів, висновків та додатків.

У вступі надано загальну характеристику роботи, виконано оцінку сучасного стану проблеми, обґрунтовано актуальність напрямку дослідження, сформульовано мету і задачі дослідження, показано наукову новизну отриманих результатів і практичну цінність роботи, наведено відомості про апробацію результатів і їх впровадження.

У першому розділі розглянуто основні способи математичного моделювання, які можуть бути використані при моделюванні оцінки якості програмного забезпечення; наведені теоретичні засади щодо побудови математичної моделі оцінки якості ПЗ.

У другому розділі сформульовано основні методичні підходи до розроблення математичної моделі якості на основі активностей (Activity-based quality model). Запропоновано алгоритм побудови моделі оцінки якості програмного забезпечення.

У третьому розділі проаналізовано метрики вихідного коду програмного забезпечення, що можуть бути використані для побудови математичної моделі оцінки якості вихідного коду програмного забезпечення. Розглянуто математичні методи системного аналізу текстових даних, що представляють

собою вихідний код програмного забезпечення, з позиції системного підходу для виявлення інтегральних характеристик вихідного коду і їх візуалізації.

У четвертому розділі досліджено проблеми з аналізом і обробленням метрик вихідного коду для оцінки якості ПЗ; побудовано математичну модель оцінки якості ПЗ; запропоновано програмну реалізацію математичної моделі оцінки якості вихідного коду ПЗ.

У висновках наведено отримані результати роботи.

Додатки містять: алгоритм побудови математичної моделі оцінки якості, що базується на активностях; порівняльні діаграми ефективності отриманих результатів; часові характеристики роботи програмного модуля аналізу параметрів математичної моделі оцінки якості ПЗ.

Робота виконана на 98 аркушах, містить п'ять додатки та посилання на список використаних літературних джерел з 49 найменувань. У роботі наведено 15 рисунків та 5 таблиць.

Ключові слова: математична модель, байесова мережа, метрика вихідного коду, якість програмного забезпечення.

РЕФЕРАТ

Актуальность темы. Несмотря на стремительное и динамичное развитие IT-индустрии, а также ее успехи по всему миру, дисциплина разработки ПО все еще не сформировалась как зрелая техническая дисциплина. Поскольку этап планирования и оценивания необходимого бюджета и сроков есть одним из наиболее принципиальных в процессе разработки ПО, то, учитывая, сложность и затратность этого процесса, увеличение точности оценок при принятии управленческих решений, является актуальным.

Объектом исследования есть процессы формализации и математической обработки текстовых данных, которые представляют исходный код программного обеспечения.

Предметом исследования является математическая модель, которая характеризует качество исходного кода программного обеспечения.

Цель работы: разработка и исследование математической модели оценки качество программного обеспечения и специальных методов математического моделирования сложных системных объектов исходного кода ПО для обнаружения интегральных свойств, характеризующих систему как единое целое.

Методы исследования. В работе используются методы математического моделирования, методы системного анализа, численные методы.

Научная новизна состоит в следующем:

- разработано новую математическую модель оценки качества ПО, которая отличается от существующих наличием большого количества метрик исходного кода, возможностью наполнения экспертными знаниями, которые подлежат моделированию, и таким образом позволяет выполнить более точную оценку качества ПО.

Практическая ценность полученных у работе результатов состоит в том, что предложенная модель и разработанные программные средства позволяют получить полную картину взаимосвязей и взаимообусловленность процессов и фактов, которые влияют на качество исходного кода ПО. Обнаружив причинно-следственные связи, можно дать обоснованную оценку качества ПО.

Апробация работы. Основные положения и результаты работы были представлены и обсуждены на научной конференции магистрантов и аспирантов «Прикладная математика и компьютинг» ПМК-2015 (Киев, 15-17 апреля 2015 г.) и опубликованы в сборнике работ XVII Международной научно-технической конференции SAIT 2015.

Структура и объем работы. Магистерская диссертация состоит из вступления, четырех разделов, выводов и приложений.

Во вступлении дана общая характеристика работы, сделана оценка современного состояния проблемы, обоснована актуальность области исследования, сформулированы цель и задачи исследования, показана научная новизна полученных результатов и практическая ценность работы, приведены сведенья об апробации результатов и их внедрении.

В первом разделе рассмотрены основные способы математического моделирования, которые могут быть использованы при моделировании оценки качества программного обеспечения; приведены теоретические основы построения математической модели оценки качества ПО.

Во втором разделе сформулированы основные методические подходы к разработке математической модели качества на основе активностей (Activity-based quality model). Предложен алгоритм построения модели оценки качества программного обеспечения.

В третьем разделе проанализированы метрики исходного кода программного обеспечения, которые могут быть использованы для построения

математической модели оценки качества программного обеспечения. Рассмотрены математические методы системного анализа текстовых данных, которые представляют собой исходный код программного обеспечения, с позиции системного подхода для выявления интегральных характеристик исходного кода и их визуализации.

В четвертом разделе исследованы проблемы анализа и обработки метрик исходного кода для оценки качества ПО; предложена программная реализация математической модели оценки качества исходного кода.

В выводах приведены полученные результаты работы.

Приложения содержат: алгоритм построения математической модели оценки качества, которая основана на активностях; непосредственно построенную модель оценки качества, сравнительные диаграммы эффективности полученных результатов.

Работа представлена на 98 страницах, содержит пять приложения и ссылки на список использованных литературных источников из 49 наименований. В работе приведены 15 рисунков и 5 таблиц.

Ключевые слова: математическая модель, байесовская сеть, метрика исходного кода, качество программного обеспечения.

ABSTRACT

Background. Despite the rapid and dynamic development of the IT industry and its success around the world, the discipline of software development has not yet formed as a mature engineering discipline. Since the stage of planning and evaluation of required budget and timetable is one of the most fundamental in the process of software development, so, given the complexity and costliness process, increase the accuracy of estimations while making management decisions is reasonable.

The object of research is the process of formalization and mathematical processing of text data that represents the software's source code.

The subject of study is a mathematical model that describes the quality of the source code of the software.

Objective: Research and development of mathematical models for evaluating software quality and special methods of mathematical modeling of complex system objects from software's source code integrated to identify properties that characterize the system as a whole.

Methods. In this paper, methods of mathematical modeling, system analysis methods and numerical methods are used.

Scientific novelty is as follows:

- Developed a new mathematical model for evaluating the software's quality. This model is different from the existing ones because of taking into account large number of software metrics, ability to be filled by expert knowledge. It allows making more accurate assessments of software's quality.

The practical value of obtained results is that the proposed model and developed tools allow getting a picture of relationships and interdependence of

processes and facts that affect the quality of the software's source code. Having established causal relationships, a thorough evaluation of quality software can be given.

Testing of work. Substantive provisions and results were presented and discussed at a scientific conference undergraduates and graduate students «Applied Mathematics and Computing» AMC-2015 (Kyiv, 15-17 April 2015) and published in the Proceedings of XVII International Scientific Conference «SAIT 2015».

The structure and scope of work. Master thesis consists of introduction, four chapters, conclusions and appendixes.

The introduction provides the general characteristics of the work, the estimation of the current state of art, the urgency of research in particular direction, purpose and objectives are formulated, the scientific novelty of the results and the practical value of work formulated described as well, information about testing results and their implementation are provided.

In the first section consists of review on the basic methods of mathematical modeling, which can be used in modeling quality assessment software. The theoretical foundations to build a mathematical model of quality assessment software described as well.

The second section formulates basic methodological approaches to the development of a mathematical model based on activity-based quality model. The algorithm for constructing such models of software's quality assessment is proposed.

The third section analyzes the software metrics that can be used to build a mathematical model of software quality assessment. The mathematical methods of system analysis that apply to text data which represents the source code of software, are reviewed from a position of systematic approach to identify the integrated characteristics of source code and their visualizations.

In the fourth section the problems of the analysis and processing of software metrics are investigated in terms of assessing the software's quality. A mathematical model for evaluation the software's quality is constructed. Software implementation of proposed mathematical model for software quality assessment is offered as well.

The conclusions consists of the results of the work.

The appendixes provides: algorithm of construction mathematical models of quality assessment, based on the activity; comparative efficiency diagram of the results; temporal characteristics of the software module parameters analysis based on mathematical models which assess the software's quality.

Work is done on 98 sheets, contains five appendixes and references to the literature list of 49 names. In this paper 15 figures and 5 tables are given.

Keywords: mathematical model, bayesian network, software metrics, software quality.

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ІТ – інформаційні технології

ММ – математична модель

ПЗ – програмне забезпечення

ABQM – activity-based quality model (модель якості на основі активностей)

DSQI – Design Structure Quality Index (індекс якості структури дизайну)

GQM – Goal, Question, Metric (Мета, Запитання, Метрика)

LCOM4 – Lack of cohesion in methods 4th generation (відсутність зв'язності у методах, 4-е покоління)

LOC – Lines of code (кількість рядків програмного коду)

NPT – Node probability table (таблиця ймовірностей для вузла)

ВСТУП

Індустрія інформаційних технологій на сьогоднішній день є галуззю економіки, що розвивається найбільш динамічними темпами. Це стосується як світу взагалі, так і України зокрема. Унікальним фактом є те, що жодна світова криза жодного разу не зупинила розвиток IT-індустрії в Україні.

У роботі [1] проаналізовано історію та поточний стан розвитку індустрії інформаційно-телекомунікаційних технологій в Україні практично з часів незалежності. Дослідження показує, що галузь стає важливою частиною національної економіки. Впевнено можна сказати, що за ці роки Україна перетворилася на надійну, зрілу та конкурентоспроможну IT-державу, а за останні роки – найпривабливішою ціллю для аутсорсингу в Східній Європі завдяки наявності висококваліфікованих фахівців та відповідного досвіду.

Проте, незважаючи на свою важливість, галузь розробки програмного забезпечення все ще не сформувалася як зріла дисципліна як в теорії, так і на практиці. За наявності корисних результатів різноманітних наукових досліджень, дуже часто при вирішенні проблем не існує єдиного загальноприйнятого способу прийняття рішень, а поради обмежуються так званими «хорошими практиками» або «найкращими практиками». Ці практики представляють собою формалізацію унікального успішного досвіду. Вважається, що застосування такої практики в іншій діяльності здатне забезпечити досягнення цілі з відповідним рівнем ефективності.

Однією з найбільш принципових проблем у процесі розробки програмного забезпечення є етап планування і оцінювання необхідного бюджету та термінів для успішного завершення проекту. Враховуючи складність та затратність процесу розробки ПЗ, навіть невелике збільшення точності оцінки якості програми, що створюється, є доцільним. Так

недооцінка необхідних на розробку ПЗ ресурсів може призвести до лише часткового виконання проекту чи до перевищення його термінів і/або бюджету. З іншого боку, через переоцінку ресурсів проект може бути відхилено, або станеться розрив між закінченням одного проекту та початком іншого.

У літературі [2, 3] описано практичний досвід виконання проектів, починаючи від розробки вимог, планування термінів та оцінки бюджету до впровадження системи у експлуатацію. Автори описують особистий досвід керівництвом проектами, наводять приклади проблем, з якими довелося зіштовхнутися, аналізують вплив прийнятих рішень на подальший розвиток проекту та дають поради щодо прийняття рішень у тій чи іншій ситуації. Обмеженням при передачі досвіду таким чином можна вважати слабку формалізацію та обмежену область використання сформульованих порад.

Тому актуальним є напрям дослідження підтримки прийняття управлінських рішень при плануванні проектів. Метою даної роботи є створення математичної моделі оцінки якості програмного забезпечення шляхом аналізу вихідного коду ПЗ за допомогою метрик. Побудована математична модель та програмні засоби показали добрі результати моделювання у порівнянні з аналогічними програмними засобами. Основні положення і результати роботи були представлені та обговорювалися на науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2015 (Київ, 15-17 квітня 2015 р.) та опубліковані у збірнику праць XVII Міжнародної науково-технічної конференції SAIT-2015.

1 СПОСОБИ РОЗРОБКИ МОДЕЛІ ОЦІНКИ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метрики вихідного коду програмного забезпечення (ПЗ) досліджуються вже понад 40 років, але за весь цей час вони ледь проникли в індустрію розробки програмного забезпечення [4]. Враховуючи складність та трудомісткість процесу розробки ПЗ, навіть невелике збільшення точності оцінки якості програми, що створюється, є доцільним. Так недооцінка необхідних на розробку ПЗ ресурсів може призвести до лише часткового виконання проекту чи до перевищення його термінів і/або бюджету. З іншого боку, через переоцінку ресурсів проект може бути відхилено, або станеться розрив між закінченням одного проекту та початком іншого.

1.1 Постановка задачі

Переважна більшість досліджень з прогнозування якісних показників ПЗ, яке розроблюється, була виконана з використанням лінійної регресії (методу найменших квадратів). Як результат, отримувалася математична модель, представлена у вигляді рівняння. Хоча цей метод і має перевагу у вигляді простоти побудови моделі та її реалізації, але перспективними є інші, більш складні методи, що враховують природу даних, які оброблюються, їхні причинно-наслідкові зв'язки тощо.

У даному розділі ставиться та розв'язується задача порівняння сучасних способів розробки моделі оцінки якості ПЗ, які здійснюються за допомогою:

- лінійної регресії;

- нейронних мереж;
- систем нечіткої логіки;
- системи, що базуються на правилах;
- байесових мереж.

1.2 Критерії оцінювання

Критерії оцінювання [5] способів побудови математичної моделі вихідного коду пов'язуються з доцільністю застосування відповідного методу (задоволення концептуальних вимог моделювання).

Оцінювання базується на перевірці наявності бажаних атрибутів у способі моделювання.

Отже, як критерії оцінювання візьмемо наступні:

- здатність способу моделювання до самовизначення структури моделі. Наприклад, при використанні регресійного способу моделювання, потрібно чітко вказати, до яких саме змінних необхідно застосувати певний вид перетворення. З іншого боку, нейронні мережі під час навчання самостійно знаходять наближене перетворення;
- стійкість моделі до оцінки, яка полягає у здатності обробляти дані, що мають викиди, тобто такі дані, які разюче відрізняються від інших;
- здатність пояснювати результати моделювання, що важливо при прийнятті управлінських рішень;
- здатність обґрунтовувати висновки, що важливо для перевірки і розуміння виконаного процесу моделювання;

- можливість додавання нових даних без суттєвої перебудови моделі;
- здатність способу працювати зі складеними моделями;
- можливість доповнення експертними знаннями.

На практиці на вибір способу моделювання можуть також впливати такі фактори:

- рівень досвідченості команди розробників;
- кількість часу, що виділений на розробку;
- наявність експертних знань з предметної області у команди розробників;
- врахування методології розробки ПЗ.

1.3 Способи побудови математичної моделі вихідного коду ПЗ

1.3.1 Регресійна модель

Лінійний метод найменших квадратів регресійного аналізу все ще є найбільш поширеним методом, який використовується. Найбільш привабливою стороною цього методу є його простота та легкість використання у багатьох популярних пакетах прикладних програм для вирішення технічних задач та математичного моделювання.

Лінійний метод найменших квадратів оперує оцінюванням коефіцієнтів у рамках виконання такої умови:

$$\min \sum_{i=1}^{n} r_i^2,$$

де r_i дорівнює різниці між статистичними даними та передбаченням моделі для i-того спостереження. Таким чином, усі дані, прийняті до уваги, і всі

елементи мають однаковий ступінь впливу на рівняння регресії. Викиди, якщо вони містяться у наборах даних, можуть мати помітний і небажаний ефект на форму прямої лінійної регресії.

Тому при роботі з даними розробки програмного забезпечення, які часто можуть бути досить малими, а, отже, дуже чутливими до ненормальних наборів даних, які, крім всього, ще й можуть містити помилки, постає питання використання стійких методів моделювання [6]. На рис. 1 наведено приклад набору даних, де один єдиний викид може кардинально вплинути на регресійну пряму, побудовану за допомогою методу найменших квадратів [7].

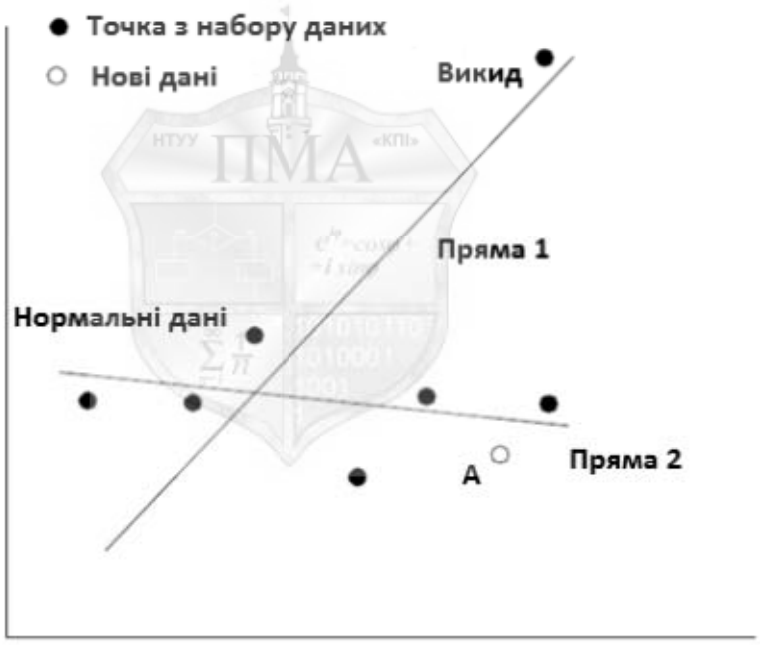


Рис. 1 – Приклад впливу викиду на модель, побудовану методом найменших квадратів

У роботах [8] та [9] наведені спроби внесення змін у спосіб побудови математичної моделі з метою уникнути такого сильного впливу викидів на результат, проте запропонований у цих роботах підхід дає лише локальний ефект.

1.3.2 Нейронні мережі

Найбільш поширеною технікою моделювання після регресійної моделі є нейронні мережі прямого розповсюдження з методом зворотного поширення помилки. Велика кількість збудованих нейронних мереж і алгоритмів навчання серед усіх різновидів нейронних мереж обмежується саме цим типом. Використання обмеженої кількості типів нейронних мереж при моделюванні пов'язують [7] з недостатнім розумінням техніки використання нейронних мереж багатьма дослідниками метрик вихідного коду ПЗ, що є зрозумілим, враховуючи величезний розвиток області нейронних мереж в останні десятиліття. У [10-11] наведено приклади застосування нейронних мереж при аналізі метрик програмного забезпечення. У [12] представлено хороший загальний вступ до використання нейронних мереж. Нейронні мережі успішно застосовуються в багатьох дослідженнях з моделювання метрик вихідного коду, а точність подекуди сягає 10% [13]. Запропонований підхід головною метою проголошував визначення точності, як найважливішої вимоги для багатьох керівників проектів. І хоча ця модель не є детермінованою, інші фактори, такі як простота використання також вважалися важливими.

На рис. 2 представлено модель нейронної мережі для метрики, що передбачає зусилля, які необхідно затратити, на розробку системи за наперед визначеним набором вимог. Мережа зображена тут – це мережа прямого розповсюдження, навчання якої можна провести за допомогою методу зворотного поширення помилки таким чином, щоб визначити ваги, які повинні мінімізувати прогностичну помилку. Після того, як ваги мережі

визначені, нові екземпляри даних можуть бути подані на вхід для оцінювання нейронною мережею необхідних зусиль.

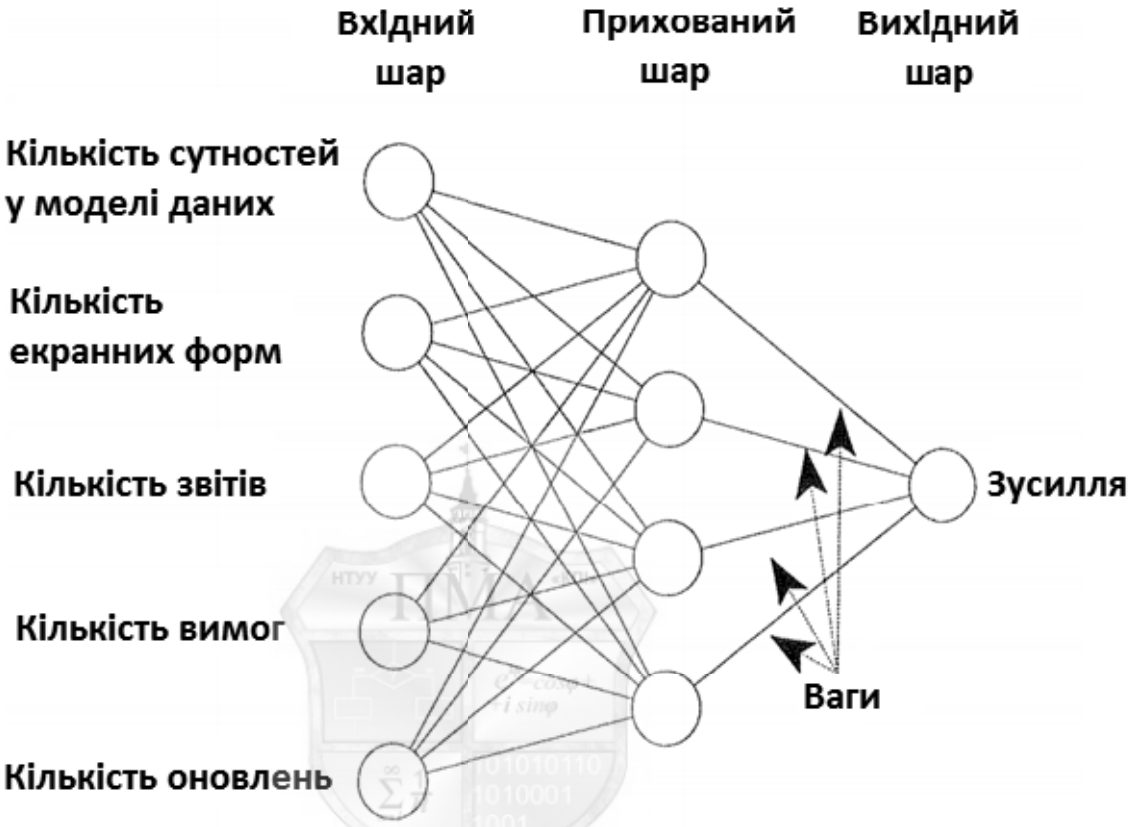


Рис. 2 – Нейронна мережа для передбачення зусиль на розробку ПЗ за заданими вимогами

На першому етапі створення нейронної мережі прямого розповсюдження з методом зворотного поширення помилки обирається прийнятна архітектура нейронів. Цей етап включає в себе вибір кількості шарів, необхідних до використання, кількість нейронів на кожному шарі та визначення зв'язків нейронів між собою. Можливими є інші шляхи побудови мережі, при яких враховується точна природа нейронів, наприклад, їх транспортна функція та параметри алгоритмів навчання. Після того, як архітектура мережі готова, відбувається її навчання, шляхом подання наборів

даних на вхід, та правильних відповідей на вихід. Як і при використанні будь-якого способу моделювання, що базується на застосуванні емпіричних даних для побудови моделі, усі дані повинні бути розділені з метою верифікації та валідації. Нейронна мережа вчиться регулювати свої ваги, щоб зменшити різницю між заздалегідь визначеним результатом виходу та фактичним результатом моделювання. Процес навчання продовжується до дих пір, поки здатність мережі до узагальнення, що вимірюється здатністю до передбачення очікуваного результату, не стане оптимальною. Це означає, що навчання мережі має бути зупинене перед тим, як вона вивчить всі дані повністю і стане перетренованою, втративши, таким чином, свою здатність до узагальнення. Як правило, при тестування випробовується велика кількість різних варіантів архітектур і з усіх варіантів обирається той, що найкраще здатен до узагальнення на етапі валідації.

Недоліком використання нейронних мереж є те, що вони функціонують як «чорні ящики», і не можуть надати жодної інформації щодо того, яким чином досягається отриманий результат на виході. Здатність генерувати пояснення є важливою у випадку необхідності отримання визнання та прийняття результатів користувачем при використанні способу моделювання.

1.3.3 Системи нечіткої логіки

Використання систем нечіткої логіки не є поширеним при моделюванні програмного забезпечення і згадується лише у декількох тематичних публікаціях [15, 16], що викликає подив, враховуючи їх широке застосування в інших областях.

У [17] надано загальний вступ до систем нечіткої логіки. Система нечіткої логіки представляє собою відображення між лінгвістичними термінами, такими як «дуже мало», на прив'язані до них змінні. Таким чином, вхідними даними нечіткої системи можуть бути як числові, так і символьні дані. Це ж стосується і вихідних даних.

У [18, 19] розглянуто різні типи систем нечіткої логіки і показано, що вони діють як універсальні «апроксиматори» у тому ж самому значенні, що і нейронні мережі. Хоча існують різні типи систем нечіткої логіки, всі вони складаються з трьох основних компонентів:

1) функція приналежності, яка відображає, наскільки задане числове значення конкретної змінної відповідає терміну, який розглядається;
2) база правил, яка може бути створена експертами на основі їхнього розуміння взаємозв'язків у предметній області, що моделюються, та покращена за допомогою різноманітних методів адаптації даних. Тут будується відображення між вхідною та вихідною функціями приналежності. Чим більший вхідний ступінь належності, тим сильніше правило спрацьовує і тим сильніше це впливає на результат у вихідній функції належності.
3) процес відновлення чіткості (дефазифікація), що може об'єднувати декілька наборів вихідних даних, які представляють собою нечіткі множини, у чітку зміну (символ чи числове значення) за ступенем приналежності.

Цей підхід продемонстрований на рис. 3. У цьому простому прикладі надані такі числові значення вхідних даних:

- розмір моделі даних – 30;
- кількість екранних форм – 26;
- розмір моделі процесу – 74.

Зазначені числові значення відображені на графіці відповідних функцій приналежності разом з кривими, що визначають безпосередньо ступінь приналежності числового значення до відповідного терміну. Конкретно для використаного типу систем нечіткої логіки, цей крок називається фазифікацією. Ступінь приналежності визначає, яку саме вагу присвоїти правилам, що задіяні у процесі виводу. Результати кожного з правил потім об'єднуються та дефазифікуються у єдине чітке значення – 254.

Найбільш очевидною перевагою систем нечіткої логіки є те, що за допомогою лінгвістичного відображення можна створити інтуїтивно зрозумілі моделі, які кожна людина, навіть за відсутності спеціальної підготовки, може розуміти і, за необхідності, критикувати. З іншого боку, системи нечіткої логіки страждають від деяких обмежень, включаючи складність визначення системи з високим ступенем точності разом з підтримкою високого ступеня осмисленості (зазвичай, більша точність потребує більшої кількості правил, а це призводить до більшої складності та меншої зрозумілості системи).

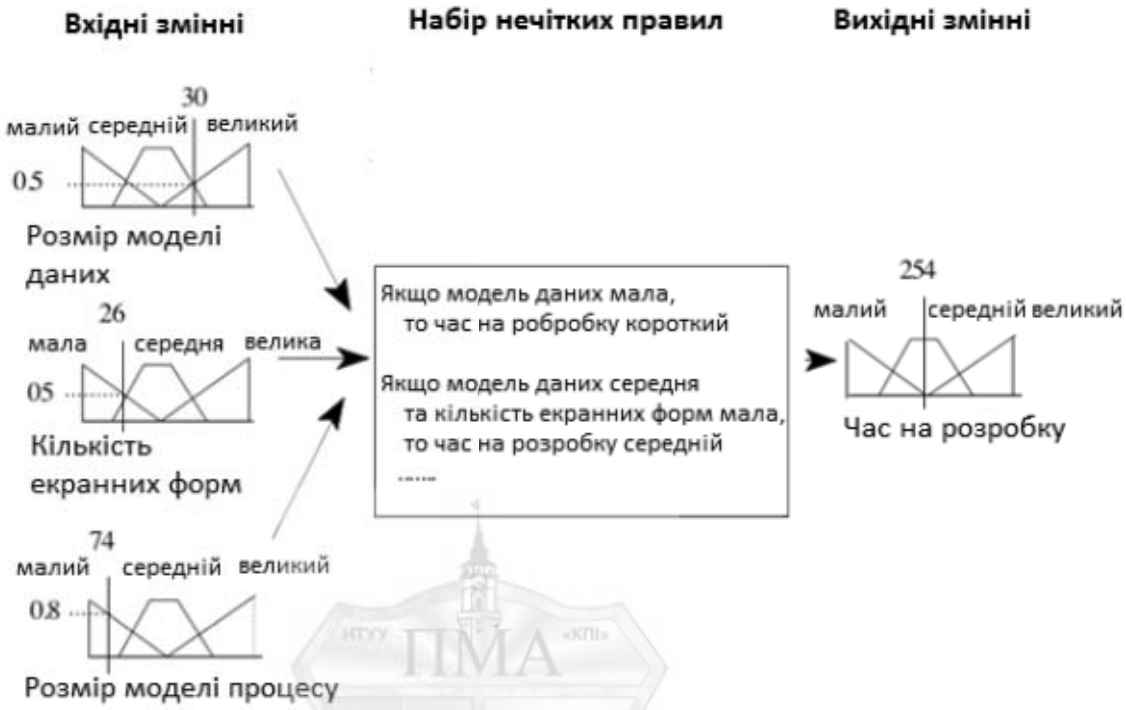


Рис. 3 – Система нечіткої логіки для оцінки тривалості розробки програмного забезпечення

1.3.4 Системи, що базуються на правилах

Системи, що базуються на правилах, для моделювання розробки програмного забезпечення були використані лише в декількох випадках [20, 21]. Взагалі, системи, що базуються на правилах, є підмножиною систем нечіткої логіки. Тобто, будь-яка модель, побудована за допомогою системи, що базується на чітких правилах, може бути так само успішно побудована і за допомогою систем нечіткої логіки. З цієї причини можна вважати, що чіткі системи є надлишковими і зайвими. Тим не менше, більша простота систем на

чітких правилах може розглядатися як приваблива особливість, особливо там, де залучена велика кількість вхідних даних.

Системи, що базуються на правилах, організовані навколо множини правил, які активуються за допомогою фактів, присутніх у робочій пам'яті. Це, в свою чергу, може активувати інші факти, як показано на рис. 4. Таким чином, рухаючись ланцюжком, одні правила можуть активувати інші.

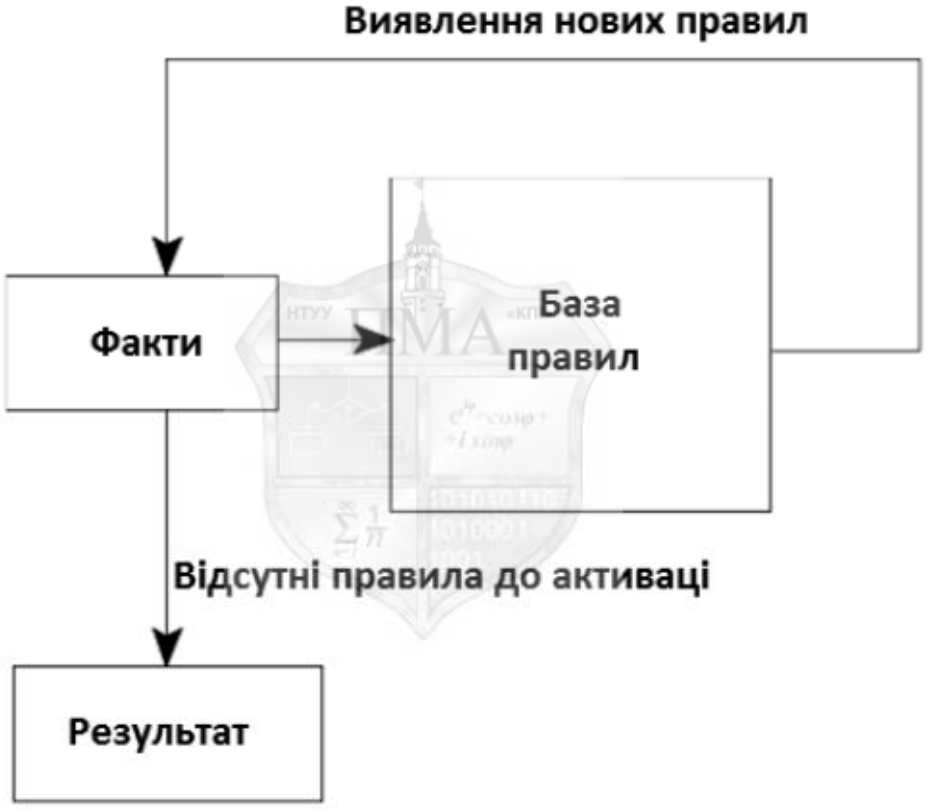


Рис. 4 – Узагальнена структура систем, що базуються на правилах

Такий ланцюг з розроблених правил, наприклад, за розміром модуля може визначити величину ступеня ризику знайти в ньому помилку:

ЯКЩО розмір модуля > 40 LOC або

ЯКЩО розмір модуля > 20 LOC ТА час на розробку > 2 год.

ТОДІ ступінь ризику цього модуля є високою.

Недоліком систем, що базуються на чітких правилах, порівняно з системами нечіткої логіки, є те, що усі передумови і наслідки повинні обов'язково бути у стані «так» або «ні», істинними чи хибними у класичному розумінні, без будь-якого ступеня належності. Це може призвести до проблеми, коли, наприклад, до системи, яка наведена у прикладі, застосувати два модулі, з розмірами 20 та 21 LOC (lines of code, кількість рядків програмного коду) відповідно. Обидва модулі дуже схожі, потребують приблизно 4 години на розробку, але у ланцюгу вони реагують на різні правила та проходять через різні шляхи до різних результатів.

1.3.5 Байесові мережі

Байесові мережі, або байесові мережі довіри, – це спосіб моделювання з представленням причинно-наслідкових зв'язків, що базується на байесовському виведенні. Мережі представляються у вигляді направленого ациклічного графа з вузлами, які містять ймовірнісні змінні, та дуги для направлення зв'язків між вершинами. Ця модель графа представляє взаємозв'язки лише абстрактно. Наприклад [22], на рис. 5 представлено простий приклад графа з трьома вершинами:

- «Складність вихідного коду»;
- «Необхідні зусилля на тестування»;
- «Кількість відмов».

У даному прикладі «Складність вихідного коду» впливає на «Необхідні зусилля для тестування» та «Кількість відмов». «Необхідні зусилля для тестування» також впливають на «Кількість відмов».

Рис. 5 – Приклад простої байесової мережі

Для кожного вузла, або змінної, задається відповідна таблиця вірогідності. Такі таблиці визначають взаємозв'язки і ступінь невизначеності цих змінних. Зазвичай, змінні є дискретними з заданою кількістю можливих станів. Для кожного стану задається ймовірність, з якою змінна може перебувати у цьому стані. Якщо вузол має предка, що впливає на поточний вузол, таблиця також визначає рівень впливу предка залежно від стану, у якому цей предок перебуває. У таблиці 2 наведено приклад таблиці ймовірності для змінної «Кількість відмов». Ця таблиця задає усі можливі комбінації для зазначеної змінної. Наприклад, у даній системі змінна «Кількість відмов» з ймовірністю 60% перебуває у стані «>100», коли обидві змінні «Складність коду» та «Необхідні зусилля для тестування» перебувають у стані «низький». З такою ж ймовірністю 60% значення «<100» змінної «Кількість відмов» можливе за умови, коли «Необхідні зусилля на

тестування» знаходяться у стані «високі», а «Складність коду» у стані «низька».

Таблиця 1 – Приклад таблиці ймовірностей для змінної «Кількість відмов» з двома станами та двома предками

Зусилля на тестування	Низькі			Високі		
Складність коду	Низька	Середня	Висока	Низька	Середня	Висока
<100	0.4	0.3	0.2	0.6	0.55	0.5
>100	0.6	0.7	0.8	0.4	0.45	0.5

Висновки

Результати порівняння моделей оцінки якості ПЗ наведені у таблиці 2. Клітинка, помічена чорним кольором, якщо даний метод моделювання задовольняє даний критерій, сірим кольором – якщо задовольняє частково, білим – якщо не задовольняє.

1) Регресійна модель

Регресійний метод доцільно використовувати для створення простих моделей з невеликим числом змінних, взаємозв'язки між якими є лінійними, або які можна звести до лінійних за допомогою перетворень.

Таблиця 2 – Порівняння способів побудови математичної моделі вихідного коду з точки зору можливостей моделювання

Спосіб моделювання	Регресійна модель	Нейронні мережі	Системи нечіткої логіки	Системи, що базуються на правилах	Байесові мережі
Визначення власної структури					
Стійкість оцінки					
Пояснення міркувань					
Додавання даних без перебудови моделі					
Обґрунтування висновків					
Об’єднання складних моделей					
Наповнення експертними знаннями					

2) Нейронні мережі

Нейронні мережі варто використовувати у системах, де точність метрик є набагато важливішою характеристикою, ніж розуміння відносин і причинно-

наслідкових зв'язків, тобто для моделювання систем, що можуть сильно відрізнятися одна від одної. До головного недоліку при виборі даного методу моделювання слід віднести те, що результати не можуть бути обґрунтованими, а тому рівень довіри до них знижується.

3. Системи нечіткої логіки

Системи нечіткої логіки варто застосовувати у випадках, коли доступна невелика вибірка даних, чи коли вони не є повними, бо такі системи дозволяють уникнути сильного зв'язку між кількісними показниками на вході та точністю оцінок у результаті.

4. Системи, що базуються на правилах

Системи, що базуються на правилах, рекомендується використовувати у таких ситуаціях, коли практично відсутня стохастична поведінка.

5. Байесові мережі

Байесові мережі варто використовувати у випадках, коли необхідно приймати рішення в умовах невизначеності. На математичній основі байесівської ймовірності можна сформулювати експертні уявлення про взаємозалежність між різними змінними та послідовно, через мережу, поширити вплив значень вихідних метрик на результат та рівень його вірогідності.

Отже, під час порівняльного аналізу показано, що деякі з сучасних методів розробки моделі оцінки якості ПЗ мають значний потенціал у забезпеченні надійного моделювання. Найкращим способом за даними таблиці 2 є байесові мережі. Саме тому цей спосіб і був обраний для подальших досліджень.

Однак, використання наведених методів не можна сприймати як спосіб вирішення усіх поточних проблем при аналізі кількісних показників ПЗ. Методи, представлені в першому розділі, – це просто засоби моделювання, які

мають допомогти у зборі та обробці інформації для прийняття управлінських рішень на конструктивній основі.

2. МАТЕМАТИЧНА МОДЕЛЬ ЯКОСТІ, ЩО БАЗУЄТЬСЯ НА АКТИВНОСТЯХ

За допомогою моделей якості можна у структурованому вигляді описати та передати суть якості програмного забезпечення. У даному розділі представлено опис моделей якості у загальному вигляді [22]. Також наведено приклади, як моделювання активностей та фактів дозволяє більш точно визначати якість.

2.1. Модель якості ПЗ

Для контролю за якістю ПЗ недостатньо визначити лише функціональні вимоги. Для успішного управління якістю необхідно точно визначити якість програмної системи в цілому. Модель якості має на меті описати, що мається на увазі під якістю, та вдосконалити цю концепцію у структурованому вигляді. На практиці, зазвичай, при цьому використовуються такі метрики як «кількість помилок» або високорівневий опис за стандартом ISO 9126 [23].

У цілому, є два основних способи застосування моделей якості у проектах з розробки ПЗ:

- як основа для визначення вимог до якості;
- для визначення способів забезпечення якості та вимірювання вимог до якості.

При першому способі інженери з розробки вимог для того, щоб дати визначення моделі якості, зазвичай, встановлюють обмеження на загально відомі атрибути, такі як надійність та здатність до підтримки. Але на практиці

це, зазвичай, скорочується до таких простих постулатів як «Система повинна бути легкою у підтримці».

Використання другого способу часто не є явним. Проте, інженери з забезпечення якості практично завжди вимірюють певні параметри, такі як кількість помилок, виявлених у системі протягом перевірок та випробувань. Відношення між такими вимірюваннями та атрибутами якості залишаються неясними. Причина полягає у відсутності практичних засобів до визначення таких високорівневих атрибутів. Отже, модель якості повинна бути якомога більш структурованою та деталізованою для того, щоб бути тісно інтегрованою у процес розробки програмного забезпечення.

2.2. Факти і активності

Використання моделей якості на основі активностей запропоновано [24] до застосування з метою усунення недоліків в існуючих моделях якості. Головна ідея полягає у тому, щоб уникнути використання розпливчастих високорівневих означень і замість цього розбити їх на детальні факти та визначити їхній вплив на активності, що виконуються у системі та над системою. На додаток до інформації про характеристики системи, модель якості містить важливі факти про процес розробки, команду, середовище та їхній вплив на, наприклад, технічні заходи підтримки та такі активності як «Читання коду», «Внесення змін» чи «Тестування». Наприклад, надлишкова кількість методів у вихідному коді, так звані клони, несуть негативний вплив на внесення змін у систему, оскільки зміни у таких клонах повинні бути виконані у різних місцях вихідного коду. Були побудовані конкретні моделі для оцінювання легкості внесення змін, легкості у користуванні та безпеки.

Для моделей якості на основі активностей визначена детальна метамодель для того, щоб охарактеризувати елементи моделі якості та їхні взаємозв'язки.

Метамодель має чотири найважливіші елементи:

- сутність;
- атрибут;
- вплив;
- активність.

Сутністю може бути будь-яка річ, жива чи не жива, яка може мати вплив на якість програмного забезпечення, наприклад, вихідний код функції або залучені тестувальники. Ці сутності характеризуються такими атрибутами як СТРУКТУРОВАНІСТЬ чи ВІДПОВІДНІСТЬ. Поєднання сутності та атрибуту називається фактом. Нотація [Сутність | АТРИБУТ] використовується для позначення факту. Для прикладу з клонами вихідного коду, запис [Функція | НАДЛИШКОВІСТЬ] можна використати, щоб позначити функцію як надлишкову. Такі факти можна вимірювати як автоматично, так і виявляти під час ручної перевірки. Якщо це можливо, подібні вимірювання зазначаються як факти всередині моделі якості, що базується на активностях.

Вплив факту визначається ефектом його дії. Під впливом на активності мається на увазі вся діяльність, що відбувається з системою. Наприклад, ПІДТРИМКА чи ВИКОРИСТАННЯ є високорівневими активностями. Вплив на активність може бути позитивним або негативним. Можлива ситуація, коли активність може мати вплив на факт, який також справляє вплив не неї. Але такі циклічні ситуації в метамоделі не розглядаються. Приклад з клонами вихідного коду можна завершити додаванням впливу на «Внесення змін»:

$$[\text{Функція} \mid \text{НАДЛИШКОВІСТЬ}] \xrightarrow{-} [\text{Внесення змін}].$$

Такий запис означає, якщо одна із сутностей системи Функція представлена атрибутом НАДЛИШКОВІСТЬ, то це буде мати негативний

вплив на активність «Внесення змін», тобто на модифікацію вихідного коду функції. Наступний приклад описує узгодженість ідентифікатору:

$$[\text{Ідентифікатор} \mid \text{ЦІЛІСНІСТЬ}] \overset{+}{\rightarrow} [\text{Внесення змін}].$$

Цей запис означає, що у випадку, якщо можна показати, що ідентифікатор є цілісним, то це буде мати позитивний вплив на діяльність з внесення змін у вихідний код розробником програмної системи. У самій моделі для документування використовується така інформація як текстовий опис, вихідний код, визначення способів оцінювання.

Модель не лише складається з опису фактів, що впливають на активності, але також містить інформацію про взаємозв'язок між ними. Як факти, так і активності можуть бути організовані у ієрархічну структуру. «Активність» найвищого рівня складається з таких підактивностей як «Використання», «Підтримка», «Адміністрування». Приклад такої ієрархії зображено на рис. 6. Підактивності, в свою чергу, підлягають подальшому уточненню. Наприклад, «Підтримка» може мати такі підактивності як «Читання вихідного коду» та «Внесення змін». Ці активності можуть бути отримані з існуючих стандартів [25] чи бути визначеними у процесі моделювання самостійно.

Через те, що факти являються сполукою сутності та її атрибуту, організація ієрархії є простою. Як правило, ієрархічні відносини між сутностями та атрибутами вже існують. Вершиною найвищого рівня у ієрархії на рис. 6 є Ситуація, або стан, процесу розробки ПЗ. Цією Ситуацією позначаються вершини усіх сутностей як системи, так і середовища розробки. У даному прикладі вона складається з Системи, її Середовища та Організації розробки ПЗ. Ці сутності повинні бути додатково уточнені. Наприклад, система може складатися з вихідного коду та інших виконуваних файлів.

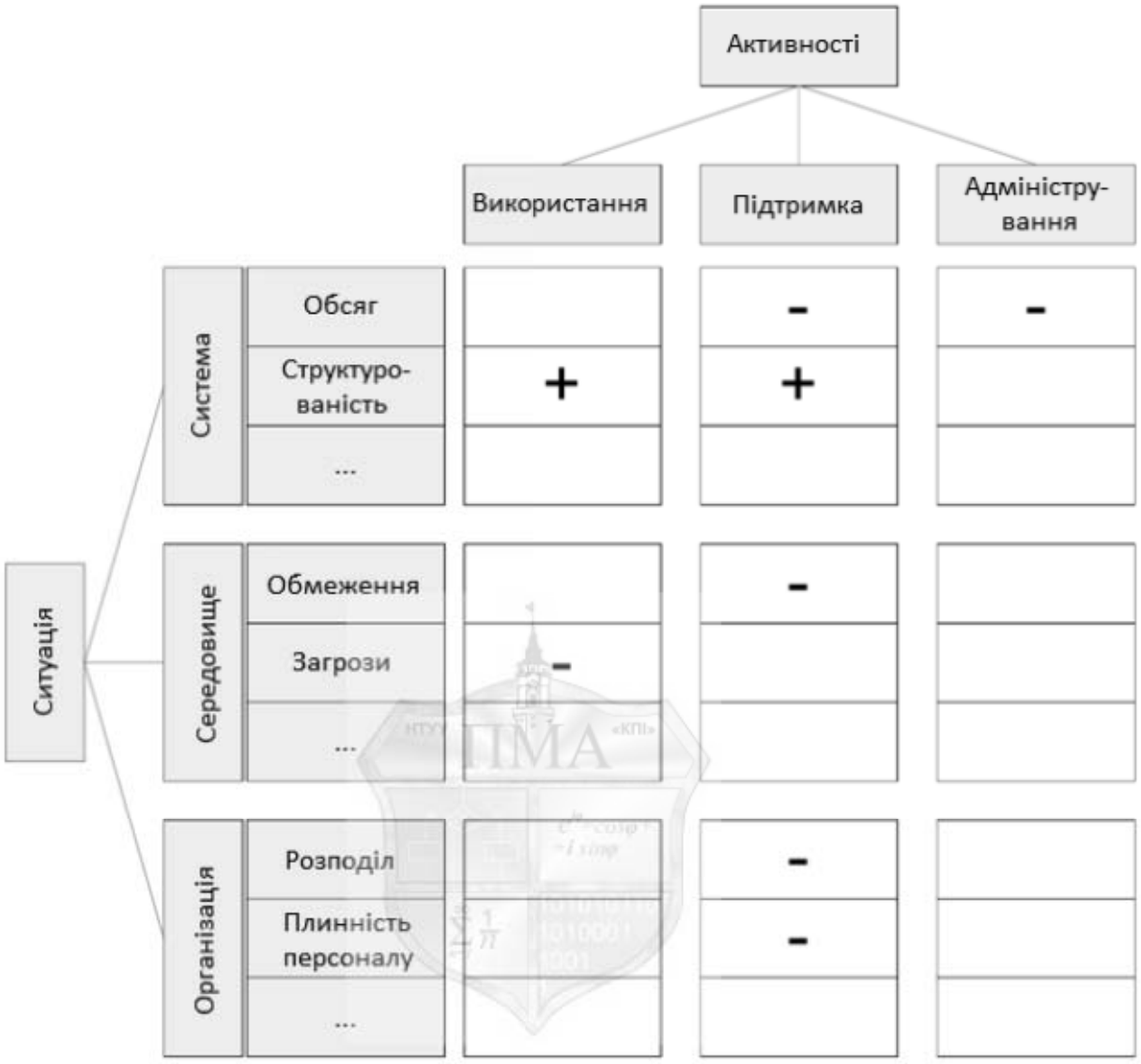


Рис. 6 – Високорівний опис моделі якості на основі активностей як матриці

Усі ці сутності можуть бути описані атрибутами. Наприклад, СТРУКТУРОВАНІСТЬ Системи. Взагалі, взаємозв'язки між сутностями можуть бути описані більш складним способом, ніж ієрархічним, але моделювання та пошук інформації, як правило, легше, якщо такі складні відносини описуються ієрархічно.

Ці дві ієрархії, дерево фактів та дерево активностей, разом із впливом фактів на активності, можуть бути зображені графічно за допомогою матриці,

так як це зроблено на рис. 6. Дерево фактів показано зліва, а дерево активностей зверху рисунку. Вплив позначається заповненням клітинок матриці, де «+» означає позитивний вплив, а «-» – негативний.

Зв'язок між фактами у дереві фактів може мати два різних значення. Сутність може бути частиною або підмножиною сутності вищого рівня. Разом з наслідуванням по ієрархії, частини сутностей верхнього рівня та їх атрибути можуть успадковуватися нащадками. Таким чином, це дозволяє більш компактно описати модель та уникнути пропусків у матриці. Наприклад, стиль іменування стосується усіх ідентифікаторів вихідного коду, незалежно від того, чи це є іменем класу, іменем файлу чи назвою змінної.

Маючи визначення усіх сутностей у метамоделі оцінки якості, можна описати, які активності потребують підтримки та вплив яких фактів повинен бути проаналізований. У термінах попереднього прикладу це б означало, що якщо потрібно підтримувати активність «Внесення змін», то для цього необхідно перевірити ідентифікатори на їх відповідність стилю іменування, узгодженості.

З іншого боку, модель якості, що базується на активностях, можна розглядати як шаблони GQM (Goal, Question, Metric – Мета, Запитання, Метрика). Активність визначає мету, а факти визначають запитання для такої мети так, щоб це можна було виміряти певною метрикою у визначеній області значень. Наприклад, метою може бути оцінити активність «Внесення змін» шляхом постановки запитання «Наскільки ідентифікатори є узгодженими?».

2.3. Алгоритм побудови байесової мережі

Для побудови байесової мережі при створенні математичної моделі оцінки якості вихідного коду ПЗ запропоновано спосіб [26], який походить від моделі якості на основі активностей. Результуюча байесова мережа складається з трьох типів вузлів:

- вузли активностей, які представляють активності на ABQM;
- вузли фактів, що відповідають фактам на ABQM;
- вузли індикаторів, що представляють метрики для активності чи факту.

Для побудови байесової мережі необхідно виконати чотири кроки, щоб відобразити всю інформацію з моделі якості на вузли байесової мережі. По-перше, потрібно ідентифікувати відповідні активності разом з індикаторами, метою яких є оцінювання чи прогнозуванні мети. По-друге, ідентифікувати вплив підактивностей на інші підактивності чи факти. Цей крок необхідно повторювати рекурсивно для усіх підактивностей. Результуючі факти моделюються разом з їхнім впливом. По-третє, відповідно до фактів додаються індикатори. По-четверте, для відображення кількісних зв'язків між вузлами мережі визначається таблиця ймовірностей. Виконавши ці кроки, байесова мережа може бути використана для моделювання шляхом встановлення значень індикаторів. Далі викладено деталізований опис чотирьох зазначених кроків.

Крок 1. Визначення активностей та індикаторів, виходячи з оцінки та прогнозування кінцевої мети. Під час виконання цього кроку використовується GQM-підхід.

Спочатку встановлюється мета оцінювання чи прогнозування – планування потреб у ресурсах на основі оцінки якості вихідного коду ПЗ. Така

мета вимагає відповідної активності (діяльності) – супроводження вихідного коду ПЗ. Ця активність уточнюється та деталізується шляхом постановки запитань та отримання відповідей на них задля досягнення мети. Наприклад, «Наскільки високими будуть ресурсні витрати на технічне обслуговування вихідного коду наступного року?». Потім обираються метрики чи індикатори, вимірювання яких дає змогу відповісти на таке запитання. Наприклад, це може бути середня кількість зусиль, необхідних для внесення змін.

Крок 2. Ідентифікація впливових підактивностей та фактів.

Тут можливі до розгляду два варіанти:

- підактивність вже ідентифікованої активності;
- вплив фактів на ідентифіковану активність.

Цей крок необхідно повторювати рекурсивно для усіх підактивностей до тих пір, поки усі факти, що мають вплив на піддерево кожної з ідентифікованих підактивностей, не будуть визначені. Відразу ж стає видимим вплив на кожну активність, а, отже, і відповідні факти. Всі активності і факти, виявлені таким чином, представляються у вигляді вузлів у байесовій мережі. Ребра додаються від підактивностей до відповідних активностей, що знаходяться вище у ієрархії, та від фактів до активностей, на які вони мають вплив.

Факти, що отримуються у результаті, в подальшому моделюються разом з кількісною величиною їхнього впливу.

На рис. 7 показано абстрактний вигляд відображення моделі якості на байесові мережу. Штриховані лінії вказують на приклади такого відображення.

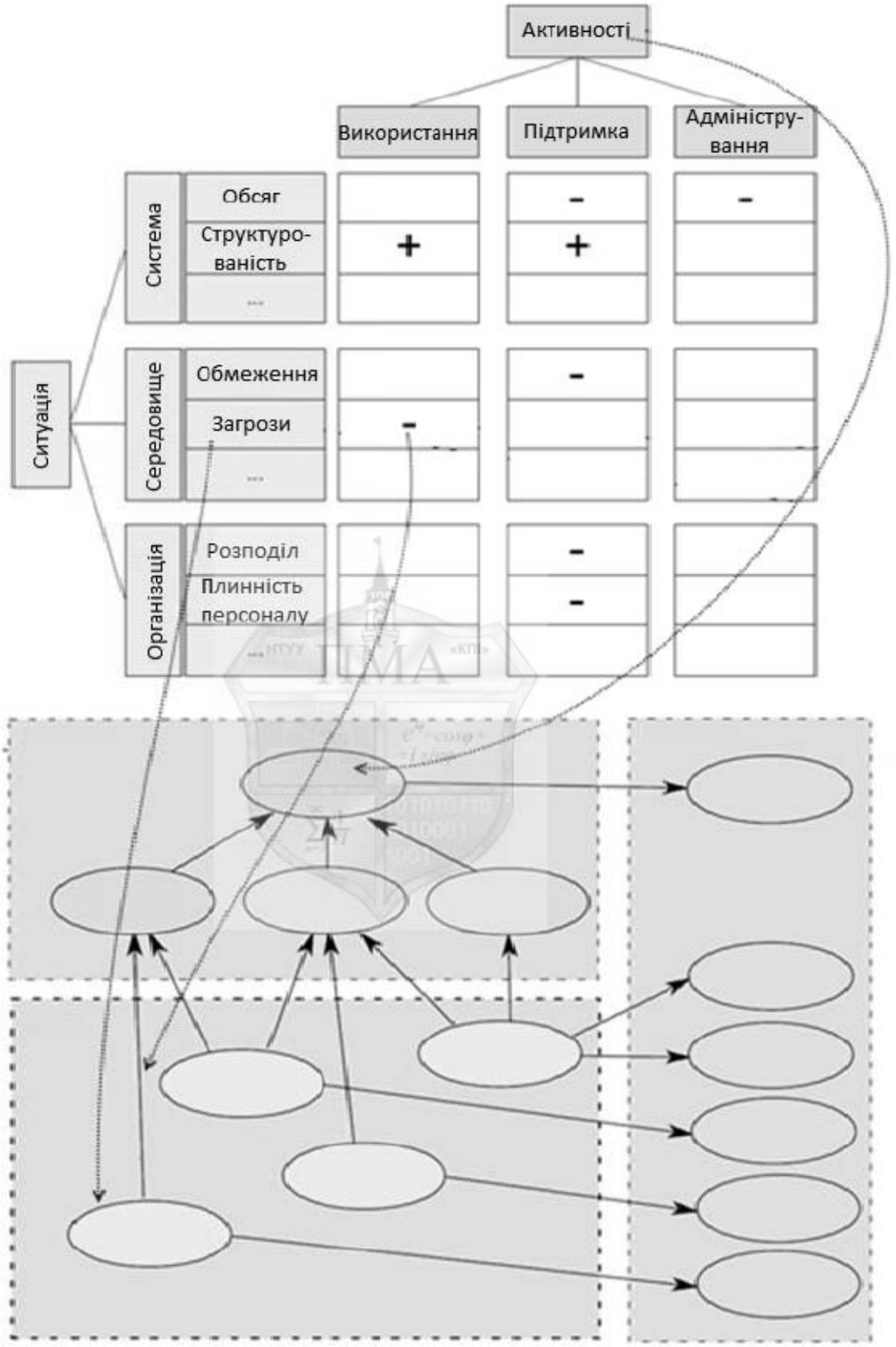


Рис. 7 – Відображення елементів моделі якості на байесову мережу

Крок 3. Додавання додаткових вузли до фактів у вигляді індикаторів для кожного факту та вузла активності, що потребують вимірювань.

На першому кроці були визначені індикатори для релевантних активностей. На другому кроці, за необхідності, можуть визначатися та застосовуватися додаткові індикатори для підактивностей. У будь-якому випадку, для кожного факту повинен бути принаймні один індикатор, що використовується при моделюванні. Це може бути вичерпна характеристика, яка лежить у межах самої моделі оцінки якості, чи самостійно побудована або запозичена з літератури вже існуюча метрика. Індикатор не повинен обов'язково вимірюватися автоматично, ручні викладки також можуть бути включеними в модель, як і експертні знання. Ребра направляються від активностей і фактів до індикаторів, тобто індикатори залежать від фактів та активностей, оскільки індикатори є лише вираженням фактору, що лежить в основі вершини, яка описується.

Головна перевага використання ABQM, як способу побудови байесової мережі, – це наявність топологічного опису. Додатковою перевагою таких моделей є якісний опис зв'язків між різними факторами, що мають відношення до якості програмного забезпечення. Припускається, що всі залежності змодельовані, а усі інші фактори є незалежними. З одного боку, це обмежує достовірність результатів моделювання за допомогою байесової мережі обґрунтованістю та коректністю моделі якості. З іншого боку, це дозволяє відокремити процес побудови байесової мережі від обґрунтування міркувань про залежність та незалежність.

Крок 4. Заповнення байесової мережі кількісною інформацією.

Даний крок включає в себе визначення станів вузла та заповнення таблиці ймовірностей для кожного з них. Вузли фактів та активностей, як правило, моделюються як ранжовані вузли, тобто за порядковою шкалою. Найпоширеніша шкала складається зі значень «малий», «середній» та

«великий». Такий підхід має переваги в оцінці та агрегації. Це робить оцінку простіше, тому що повинні бути визначені не точні числа, а їх приблизні рівні. Насправді, такі рівні здатні відображати високу ступінь невизначеності даних. Також такі приблизні рівні легше обробляти у процесі агрегації вузлів (вгору по ієрархії дерева активностей). Нарешті, для такої скінченної кількості рівнів визначити специфікацію агрегації легше, ніж для неперервних даних.

Висновки

В даному розділі показано, що за допомогою моделей якості можна у структурованому вигляді описати та передати суть якості програмного забезпечення. Наведено приклади, які показують, що моделювання активностей та фактів дозволяє більш точно визначати якість.

Надійне кількісне оцінювання та передбачення якості ПЗ є однією з найголовніших цілей у галузі управляння якістю ПЗ. В літературі описано багато продуктивних та корисних зусиль не тільки щодо побудови моделей оцінювання та прогнозування, а й в області визначення обмежень у використанні таких моделей. Тим не менш, ці моделі ще не є тісно інтегрованими у інші діяльності з управління якістю. Модель якості, що базується на активностях, на практиці довела, що може слугувати міцним фундаментом у задачах визначення якості на детальному рівні. Проте, використання кількісного аналізу до недавнього часу було обмеженим.

3. КІЛЬКІСНІ ПОКАЗНИКИ ВИХІДНОГО КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Метрики програмного забезпечення, як предметна область для прикладних досліджень, розвивається вже більше 40 років [27], але за весь цей час вони так і не стали ключовим елементом в галузі програмної інженерії. Основною причиною цього є те, що відомі метрики вихідного коду не забезпечують виконання наступної важливої вимоги: надання кількісної інформації для підтримки прийняття управлінських рішень у ході життєвого циклу програмного забезпечення. Хороша підтримка при прийнятті рішень має на меті надання допомоги в правильній оцінці ризиків. Тим не менше, традиційні підходи у використанні метрик часто використовують регресійну модель для оцінки бюджету та передбачення кількості помилок. Такий підхід слабо задовольняє потреби керівників, які мають на меті проводити вимірювання для аналізу поточної ситуації та зниження ризиків. Майбутнє метрик вихідного коду програмного забезпечення полягає у використанні відносно простих існуючих метрик для побудови інструментів для підтримки прийняття рішень, які поєднують різні аспекти процесу розробки програмного забезпечення та його тестування. Це має дозволити керівникам проектів виконувати багато видів передбачень, оцінок та бути готовими до компромісів протягом життєвого циклу проекту. У роботі [27] запропоновано використовувати підхід, при якому обробці та аналізу підлягають ключові фактори, що зазвичай відсутні у звичайних способах використання метрик, а саме: причинно-наслідкові зв'язки, невизначеність, комбінування різних, часто суб'єктивних, свідчень. Таким чином, головний напрямок дослідження метрик ПЗ лежить у моделювання причинно-наслідкових зв'язків (у розділі 1

для цього запропоновано використовувати байесові мережі) та застосуванні багатокритеріальних методів прийняття рішень.

3.1. Постановка задачі

Загальноприйнятим є факт, що важливим компонентом вдосконалення процесу є здатність до вимірювання цього процесу. Враховуючи центральну роль, яку відіграє розробка ПЗ у поставці високотехнологічних рішень та повсюдне застосування інформаційних технологій, керівники проектів все більше звертають увагу на можливості поліпшення процесів у області розробки ПЗ. Така підвищена увага мала два наслідки:

1) попит стимулював розвиток нових або вдосконалення існуючих підходів до розробки ПЗ;
2) фокус на покращенні процесу розробки збільшив попит на вимірювання кількісних показників ПЗ, чи метрик, за допомогою яких можна вимірювати процес та управляти ним.

Потреба в таких метриках є особливо гострою при переході організації на нову технологію, досвіду використання якої ще не напрацьовано.

У [28] досліджено можливості поліпшення процесів розробки ПЗ за рахунок розвитку і впровадження нових метрик. Зазначається, що попередні дослідження метрик вихідного коду, як правило, піддавалися одному чи декільком типам критики. Найбільш поширеними зауваженнями є

- відсутність теоретичної основи;
- недостатність бажаних властивостей вимірювання;
- недостатня узагальненість;
- надмірна залежність імплементації від певної технології;

- надмірна трудомісткість при зборі.

Ключовою метою даного розділу є дослідження існуючих метрик вихідного коду для подальшої емпіричної перевірки узгодженого набору теоретично обґрунтованих метрик.

3.2. Вимоги до вибору метрик

Деякі дослідження [28, 29] рекомендують властивості, якими повинні володіти метрики вихідного коду, для того щоб їхня користь була найбільшою. Наприклад, у [29] встановлено, що показники вихідного коду повинні бути чутливими до відмінностей, які можуть бути поміченими ззовні, у середовищі розробки ПЗ. Вони також повинні відповідати інтуїтивним уявленням про характерні відмінності між артефактами ПЗ, що вимірюється. Більшість рекомендованих властивостей є якісними за своєю суттю, а, отже, більшість пропозицій для метрик, як правило, є неформальними у своїй оцінці.

Проте, у [30] розроблено формальний список побажань до метрик ПЗ та пораховано кількість існуючих метрик, яка відповідає цим властивостям. Наведемо деякі з них.

Властивість 1. Не жорсткість.

Для заданого об'єкту A та метрики μ може бути знайдений такий об'єкт $Б$, що

$$\mu(A) \neq \mu(Б).$$

Це означає, що всі об'єкти не можуть мати однакові значення за метрикою μ.

Властивість 2. Не єдність (поняття еквівалентності).

Можуть існувати 2 різні об'єкти A та $Б$ ($A \neq Б$), такі що

$$\mu(A) = \mu(Б).$$

Це означає, що два об'єкти можуть мати одне й те саме значення у метриці μ, тобто ці два об'єкти мають однаковий рівень складності.

Властивість 3. Нефункціональна імплементація є важливою.

Якщо є два об'єкти A та $Б$, що надають однакову функціональність (A=$Б$), то це не гарантує виконання рівності

$$\mu(A) = \mu(Б).$$

Це означає, що навіть якщо два об'єкти спроектовані для виконання однакової функціональності, то саме деталі реалізації повинні визначати значення метрики цих об'єктів.

Властивість 4. Монотонність.

Для всіх об'єктів А та Б повинні виконуватися нерівності:

$$\mu(A) \leq \mu(A + Б),$$
$$\mu(Б) \leq \mu(A + Б),$$

де $A + Б$ визначає конкатенацію об'єктів A та $Б$.

Це означає, що значення метрики для комбінації з двох об'єктів не може бути меншою за значення метрики окремо кожного з об'єктів.

Властивість 5. Не еквівалентність взаємодії.

Існують такі A, $Б$ та B, що якщо $\mu(A) = \mu(Б)$, то це ще не означає, що

$$\mu(A + B) \leq \mu(Б + B).$$

Це означає, що взаємодія між А та В може відрізнятися від взаємодії Б та В і в результаті мати різну складність для A+B та $Б$+B.

Властивість 6. Взаємодія збільшує складність.

Для будь-яких A та $Б$, виконується нерівність

$$\mu(A) + \mu(Б) \leq \mu(A + Б).$$

Це означає, що коли два об'єкти поєднуються, то від їх взаємодії результуюче значення метрики має тенденцію до збільшення.

Процес розробки архітектури ПЗ включає в себе використання розробниками неявних власних ідей та уявлень про складність. З цієї точки зору, в даному розділі у формальному вигляді наведені деякі з емпіричних або неявних відносин між об'єктами. Саме на основі цих властивостей рекомендується обирати метрики для збору кількісних показників вихідного коду ПЗ. Окремо варто зазначити незалежність запропонованих властивостей від мови програмування.

3.3. Метрики аналізу

3.3.1. Індекс якості структури дизайну

Індекс якості структури дизайну (Design Structure Quality Index, DSQI) є метрикою архітектурного дизайну, що використовується для оцінки дизайну ПЗ та ефективності його модулів [31]. Метрика була розроблена Командуванням системами Повітряних сил США (United States Air Force Systems Command).

Результатом обрахунків є число від 0 до 1. Чим ближче значення до 1, тим більшою вважається якість дизайну. Краще всього її використовувати як основу для порівняння з попередніми успішними проектами.

Онлайн-версія калькулятора та методика обрахунку метрики доступна за посиланням [32].

3.3.2. Середній розмір модулів

Кількість рядків коду – це метрика ПЗ, що використовується для вимірювання його об'єму за допомогою підрахунку кількості рядків у тексті вихідного коду.

Результати, що отримуються за допомогою даної метрики, часто є суперечливими, особливо при некоректному використанні. Хоча зазначається [33], що дана метрика добре корелює з трудовими затратами – програмне забезпечення з великою кількістю коду потребує більше часу на розробку та підтримку. Через це використання даної метрики вважається виправданим. Хоча з іншого боку, кореляція з функціональністю вже не є такою очевидною. Досвідчені розробники ПЗ віддають перевагу написанню меншої кількості коду для вирішення тієї самої задачі. І якщо при оцінці продуктивності достатньо великої команди різниця у класі розробників може нівелюватися, то використання даної метрики для оцінки продуктивності окремого працівника вже представляється неадекватною.

Відносно нещодавно з'явився ще один аспект даної проблеми – різниця між кодом, що написаний вручну, та кодом, що згенерований автоматично. Сучасні засоби розробки досить часто надають можливість автоматично створювати великі об'єми коду за допомогою лише декількох ручних операцій. Найбільш яскравим представником таких систем є засоби візуальної розробки графічного користувацького інтерфейсу. Об'єм роботи, що витрачений на створення такого коду, ніяк не може бути порівняний з об'ємом роботи по написанню, наприклад, драйвера пристрою.

3.4. Метрики гарантування якості

3.4.1. Частка коду, покритого тестами

Покриття коду – метрика, що використовується при тестуванні ПЗ і показує наскільки вихідний код програми є протестованим. Техніка покриття вихідного коду тестами була однією з найперших методик, створених для систематичного тестування ПЗ. Вперше запропонована у 1963 році [34].

Для вимірювання частки коду, покритого тестами, використовується один або декілька критеріїв покриття. Наприклад:

- покриття функцій: чи кожна функція у системі була виконана;
- покриття операторів: чи кожен рядок вихідного коду був виконаний і протестований;
- покриття умов: чи кожна точка умови (обчислення виразу на істинність чи хибність) була виконана і протестована;
- покриття шляхів: чи всі можливі шляхи через задану частину коду є протестованими.

Для програм з особливими вимогами до безпеки часто ставить вимога демонстрації того, що 100% коду є покрите тестами принаймні за одним з критеріїв. Деякі з критеріїв покриття пов'язані між собою. Наприклад, покриття шляхів включає в себе і покриття умов, і покриття операторів. Хоча покриття операторів не включає в себе покриття умов.

Щодо практичного використання, то зазвичай вихідний код супроводжується тестами, які регулярно виконуються. Отриманий звіт аналізується з метою виявлення області коду, що не виконувалася. Згодом набір тестів оновлюється для непокритих областей. Мета полягає у тому, щоб отримати набір тестів для регресійного тестування, які ретельно перевіряють весь вихідний код.

Ступінь покриття коду зазвичай представляють у вигляді відсотків. Наприклад, «59% коду є протестованим». Звичайно, зміст такої фрази залежить від того, який критерій використовується. Наприклад, 59% покриття шляхів – кращий результат ніж 59% покриття операторів. Також відкритим залишається питання покриття коду та якості тестового набору даних.

3.4.2. Цикломатична складність

Цикломатична складність — метрика ПЗ, розроблена Томасом Мак Кабе [34]. Використовується для оцінки складності програм. Обчислює кількість лінійно незалежних шляхів у алгоритмі роботи програми на основі її вихідного коду.

Цикломатична складність обчислюється на основі графу, що відображає цикл роботи програми. Вершинам графа зіставляють команди програми. Ребро сполучає дві вершини якщо друга команда може бути виконана відразу після першої.

Цикломатична складність відрізку вихідного коду — кількість лінійно незалежних шляхів у вихідному коді. Наприклад, якщо вихідний код не містить місць прийняття рішень таких як IF-тверджень або FOR-циклів, складність дорівнює 1 через наявність лише одного шляху у вихідному коді. Якщо код містить один IF, тоді в вихідному коді наявні два шляхи: один, якщо твердження умови IF оцінюється як TRUE, і другий, якщо як FALSE.

Математично, цикломатична складність програми описується за допомогою орієнтованого графа, утвореного базовими блоками програми, з ребрами між двома базовими блоками, якщо керування може бути передане

від першого до другого (граф потоку керування програми). Тоді складність визначається як:

$$M = E - N + 2P,$$

де

M – цикломатична складність;

E – кількість ребер у графі;

N – кількість вершин у графі;

P – кількість компонент зв'язності.

На рис. 7 представлено приклад графу виконання простої програми. Виконання починається у червоній вершині, потім входить у цикл (група з трьох вершин безпосередньо за червоною вершиною). Після виходу з циклу знаходиться умовний оператор (наступна група вершин після циклу). Виконання програми завершується у синій вершині.

Граф, зображений на рис. 7, має такі властивості:

- кількість ребер – 9;
- кількість вершин – 8;
- кількість компонент зв'язності – 1.

Отже, цикломатична складність такої програми дорівнює 3.

Взагалі, у [35] доведено, що цикломатична складність програми з однією точкою входу та однією точкою виходу дорівнює кількості місць прийняття рішень (IF- або FOR-операторів) плюс один.

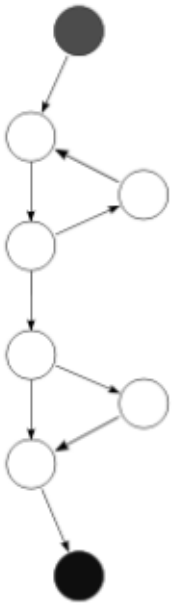


Рис. 7 – Приклад графа потоку виконання програми

3.5. Метрики імплементації

3.5.1. LCOM4

Принцип єдиного обов'язку в об'єктно-орієнтованому програмуванні говорить про те, що клас не повинен мати більше однієї причини для внесення змін у нього. Пов'язаність (cohesion) – це ступінь, з якою методи одного класу пов'язані між собою. Наприклад, якщо дві функції класу не використовують одні й ті ж самі атрибути цього класу або одні й ті ж самі методи, це означає, що у них немає нічого спільного і, можливо, вони не повинні відноситися до одного класу (згідно принципу єдиного обов'язку). Іншими словами, такий клас можна розділити на декілька нових окремих класів задля покращення модульності вихідного коду на рівні класів.

На рис. 8 зображено приклад класу, значення метрики LCOM4 [35] якого дорівнює 2. Клас на рис. 8 має 5 методів: A, B, C, D, E та 2 поля даних x та y. Як видно, методи A та B, що використовують поле x, повністю відокремлені від функцій C, D та E, які використовують атрибут y. Тому цей клас варто розділити відповідно на дві частини, а, отже, значення метрики при цьому дорівнює 2.

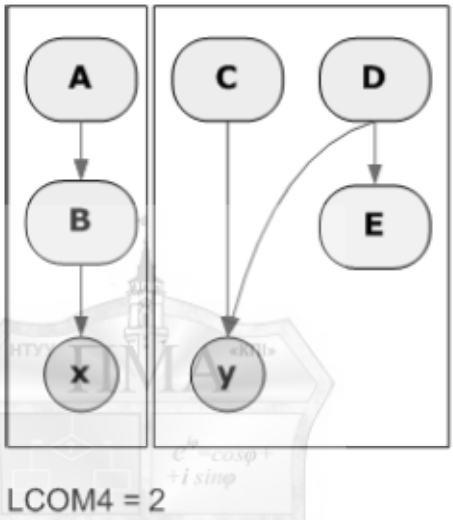
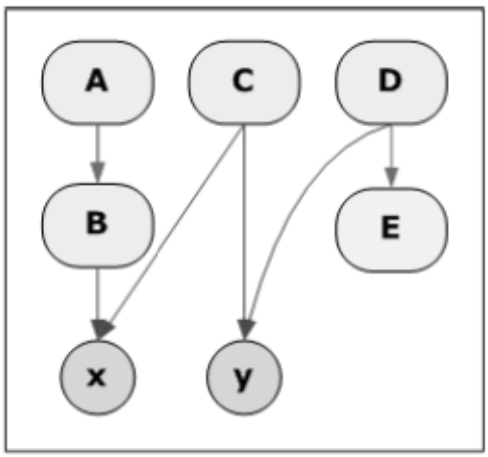


Рис. 8 – Приклад класу, що не відповідає принципу єдиного обов'язку

На рис. 9 зображено клас, який складається з таких же компонентів, як і клас на рис. 8, але які взаємодіють по-іншому. У цьому випадку, клас не можна розділити на дві підмножини, що не використовують одні і ті ж атрибути чи функції. Такий клас відповідає принципу єдиного обов'язку і тому його значення за метрикою LCOM4 дорівнює 1.

У загальному випадку, клас може містити 3, 4 і навіть більше підмножин, що не мають спільних атрибутів у користуванні. В такому випадку, значення їх метрик буде, відповідно, дорівнювати 3 та 4.



LCOM4 = 1

Рис. 9 – Приклад класу, що відповідає принципу єдиного обов'язку

Високе значення LCOM4 метрики зазвичай говорить про те, що клас погано придатний до використання і повинен бути реструктуризований.

3.5.2. Середня зв'язність модулів

Зв'язність – це спосіб та ступінь взаємозалежності між програмними модулями або сила взаємозв'язку між модулями. Зв'язність, зазвичай, протиставляють пов'язаності. Слабка зв'язність часто супроводжується сильною пов'язаністю і навпаки. Як метрика ПЗ зв'язність вперше описана у [37]. Слабка зв'язність часто є ознакою добре структурованої комп'ютерної системи та ознакою хорошого проекту, а у випадку комбінування з сильною пов'язаністю, відповідає загальному хорошому показнику легкості супроводження ПЗ.

Зв'язність може бути «слабкою» (чи «низькою») або «сильною» (чи «високою»). Розглянемо деякі види зв'язності у спадаючому порядку.

1. Зв’язність за вмістом.

Означає, що один модуль модифікує чи використовує внутрішні дані іншого модуля. Тобто зміна способу обробки даних у іншому модулі, вимагатиме внесення відповідних змін у перший модуль.

2. Зв’язність за спільністю даних.

Означає, що два модулі використовують одні й ті ж самі глобальні дані. Тобто зміна ресурсу передбачає внесення змін у всі модулі, що його використовують.

3. Зовнішня зв’язність.

Означає, що два модулі мають спільний нав’язаний ззовні формат даних, протокол комунікації чи інтерфейс пристрою. Зазвичай, пов’язане з взаємодією із зовнішніми інструментами чи апаратурою.

4. Зв’язність контролю.

Означає, що один модуль контролює хід роботи іншого модуля через передачу йому команд до виконання.

5. Зв’язність через структуру даних.

Означає, що модулі мають спільну складну структуру даних, проте використовують лише їх частину. У загальному випадку, ці підмножини можуть навіть не перетинатися.

6. Зв’язність даних.

Означає, що модулі обмінюються даними як параметрами. Кожен параметр є елементарним і представляє собою єдине ціле.

7. Зв’язність через повідомлення.

Найслабший тип зв’язності. Досягається за допомогою децентралізації станів об’єктів. Взаємодія компонентів та обмін даними проводиться через параметри та обмін повідомленнями.

8. Відсутня зв’язність.

Означає, що модулі взагалі не взаємодіють між собою.

У [38] описуються варіанти кількісного опису зв'язності.

3.5.3. Відносна кількість коментарів

Щільність коментарів – це відношення кількості рядків, що представляють собою коментарі, до загальної кількості рядків вихідного коду. Стверджується [39], що відносна кількість коментарів є хорошою ознакою для передбачення складності чи простоти підтримки ПЗ. Також показано, що середнім показником щільності коментарів у відкритих проектах, є 19%. При цьому всьому не встановлено залежності між щільністю коментарів та такими параметрами як розмір проекту, кількість операторів, проте є залежність від віку проекту. З часом щільність коментарів знижується, а після 48 місяців, в цілому, тримається стабільною.

Висока щільність коментарів спостерігається у проектах, які використовують коментарі для автоматичної генерації документації. У таких випадках щільність може зростати аж до 75% [40]. Це, в свою чергу, також призводить до витрат часу на необхідність підтримки актуальності коментарів разом із змінами вихідного коду.

Висновки

Необхідним компонентом вдосконалення процесу розробки ПЗ є здатність до вимірювання цього процесу. У даному розділі наведено теоретичні засади щодо властивостей, якими повинні володіти метрики

вихідного коду, та, відповідно до цих вимог, обрано метрики, які рекомендується використовувати при побудові моделі оцінки якості ПЗ. Список рекомендованих метрик наведено у таблиці 3 разом із областю значення кожної з них.

Таблиця 3 – Область значення метрик вихідного коду

Назва метрики	Мін. значення	Макс. значення
DSQI	0	1
Середній розмір модуля	0	Не обмежено
Частка коду, покритого тестами	0	1
Цикломатична складність	1	Не обмежено
LCOM4	1	Не обмежено
Середня зв'язність модулів	0	1
Відносна кількість коментарів	0	1

4. ПОБУДОВА МАТЕМАТИЧНОЇ МОДЕЛІ

Процес побудови байесової мережі складається з ідентифікації важливих змінних, що повинні бути змодельовані, представлення їх як вузлів, побудови топології і визначення таблиць імовірностей для вузлів мережі. Кожен з цих кроків важливий та нетривіальний. По-перше, ідентифікація важливих змінних включає в себе припущення, що особа, яка будує модель, здатна визначити певну основу з найнеобхідніших та найважливіших змінних. Нажаль, у багатьох випадках це заздалегідь невідомо. Однією з можливостей є спочатку включити багато змінних та провести аналіз їх чутливості, щоб згодом на основі цього аналізу відсіяти найменш важливі. Створити модель, яка б описувала ситуацію повністю, як правило, не представляється можливим, бо мережа стає занадто заплутаною, складною у побудові та, що буває найчастіше, відсутні знання про взаємодію багатьох змінних.

По-друге, при побудові топології використовується припущення, що особа, яка займається створенням моделі, може приймати рішення щодо залежності чи незалежності ідентифікованих змінних. У процесі будівництва байесової мережі, особливо стосовно припущень щодо незалежності (наприклад, відсутності ребра на графі), слід докладно обґрунтовувати прийняті рішення. По-третє, проблема визначення таблиць імовірностей є загальновідомою у літературі [41]. Великою частиною цієї проблеми є те, що цей процес вимагає визначення кількісних відносин між змінними. Існують різні можливі методи для визначення такої кількості, такі як колесо ймовірності чи регресія на основі емпірично зібраних даних. Проте, усі методи мають певні недоліки.

Важливо відзначити, що кожен з цих кроків є важливим та помилка на кожному з цих етапів може мати великий вплив на кінцевий результат.

Байесові мережі та відповідні програмні засоби дозволяють легко будувати моделі та отримувати кількісні результати.

4.1. Побудова байесової мережі

У розділі 2 наведено детальний алгоритм побудови байесової мережі, а у Додатку А наведено загальну блок-схему алгоритму. Для того, щоб відобразити всю інформацію з метамоделі якості на основі активностей на байесову мережу необхідно виконати чотири кроки.

4.1.1. Визначення активностей та індикаторів

На першому кроці потрібно визначити активності та індикатори, виходячи з кінцевої мети. Оскільки кінцевою метою є отримання кількісної оцінки якості ПЗ, то це і буде головним індикатором. Головною активністю, при цьому, є супроводження вихідного коду. Сюди можна віднести додавання нового функціоналу, виправлення відомих помилок та рефакторинг вихідного коду задля підвищення його якості та легкості роботи з ним.

Під час виконання першого кроку, згідно GQM-підходу, необхідно поставити запитання «Що впливає на складність та/або тривалість вирішення задач щодо супроводження вихідного коду?». Відповідь на це запитання дозволить встановити впливові активності, що здійснюють вплив на активність «Супроводження вихідного коду». В роботі запропоновано виділити при цьому три головні активності:

1) Аналіз. Проста та зрозуміла архітектура ПЗ, виконана за допомогою новітніх засобів візуального моделювання та побудована з використанням сучасних та загальноприйнятих методологій проектування, дозволяє розробнику швидко виконати етап аналізу існуючої системи та локалізувати область пошуку існуючої проблеми або знайти вірне місце для додавання нового функціоналу.
2) Гарантування якості. Процес тестування значною мірою впливає на якість ПЗ. Слідування добре визначеному процесу гарантування якості, який може складатися як з автоматизованого, так і ручного тестування, дозволяє своєчасно виявляти помилки та підтримувати якість ПЗ на високому рівні.
3) Імплементація. Слідування сформульованим емпіричним принципам при розробці ПЗ та загальноприйнятим «хорошим практикам», використання сучасних підходів до вирішення стандартних проблем дозволяє підтримувати код зрозумілим іншим розробникам, що, в кінцевому результаті, позитивно впливає на його якість.

Візуалізація результату виконання першого кроку зображено на рис. 10.

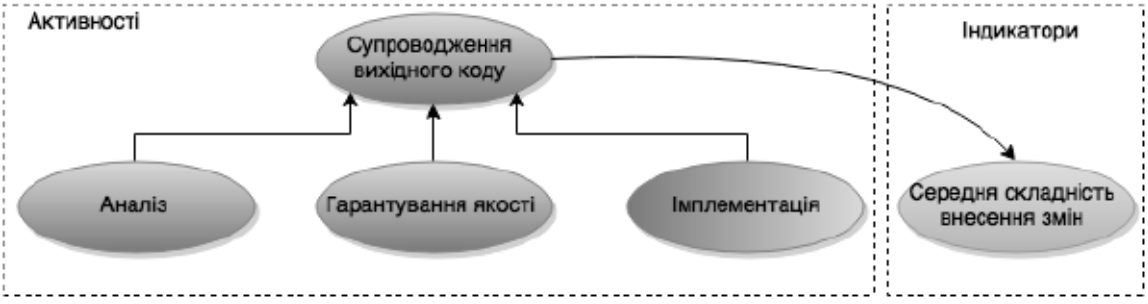


Рис. 10 – Результат виконання першого кроку

4.1.2. Ідентифікація впливових підактивностей та фактів

Головною задачею при виконанні другого кроку є ідентифікація впливових підактивностей та фактів, що, в кінцевому результаті, впливають на якість ПЗ. На першому етапі рекурсивно визначаються всі впливові підактивності, а на другому – визначеним підактивностям зіставляються факти, що їх характеризують.

1) На процес аналізу ПЗ впливає здатність до розуміння архітектури ПЗ та зручність навігації по коду. Розбиття вихідного коду на зручні для роботи модулі (файли, класи) підвищує зручність до співставлення різних частин системи під час їхньої взаємодії, тобто виявлення, потоків даних та команд, інтерфейсів та протоколів взаємодії. Отже, для активності «Аналіз» можна визначити підактивність «Розуміння», яка в свою чергу, складається з «Проектування» та «Читання коду».
2) Гарантування якості досягається шляхом автоматичного чи ручного тестування. Цей процес потребує чіткого визначення методу тестування (ручне, автоматичне, змішане), сценаріїв тестування, регулярності тестування (ручне тестування перед випуском нової версії продукту, виконання регресійних тестів після внесення змін у вихідний код).
3) Легкість внесення змін в існуючий вихідний код зумовлюється структурованістю вже існуючої імплементації, якістю розділення системи на модулі та внутрішньою узгодженістю цих модулів, способом документування (найчастіше програмний код є єдиною документацією та єдиним результатом процесу розробки).

На рис. 11 зображено результати виконання проміжного етапу другого кроку.

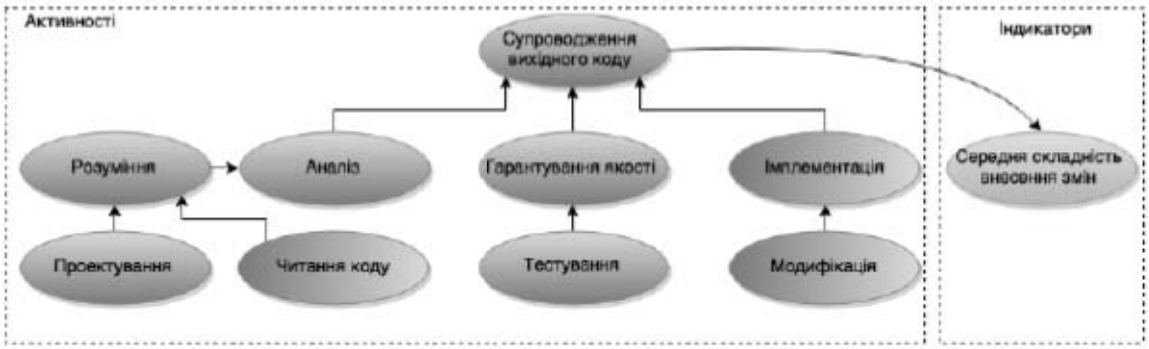


Рис. 11 – Результат визначення впливових підактивностей

На другому етапі кожній з визначених підактивностей зіставляються факти, що їх характеризують. Кожна з активностей повинна бути представлена принаймні одним фактом. Фактом у даному випадку є те, що можна представити у кількісному відношенні до кожної підактивності. З одного боку, факт одразу випливає з назви активності. Наприклад для активності «Читання коду» можливим фактом є «Обсяг модулів», який легко може бути порахований. З іншого боку, для заданої активності, факти можуть представлятися взаємопов'язаними парами. Так, наприклад, для активності «Модифікація» такими фактами будуть «Пов'язаність» та «Зв'язність».

На рис. 12 зображено результат визначення фактів для усіх активностей.

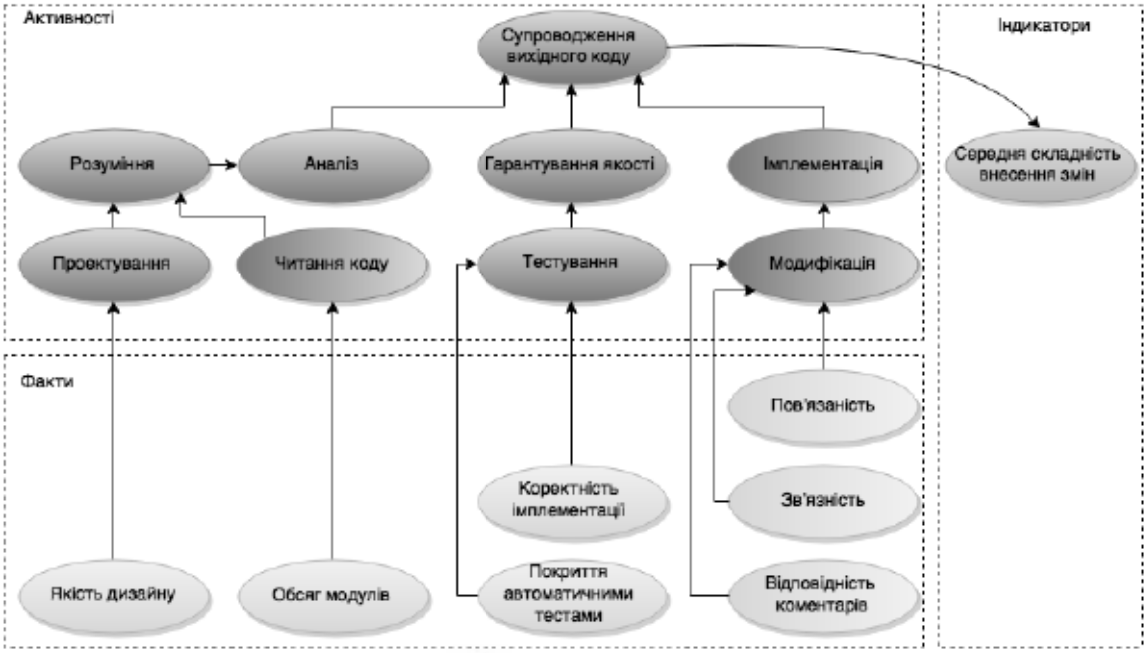


Рис. 12 – Результат визначення фактів для усіх активностей

4.1.3. Додавання індикаторів до фактів

На третьому кроці до фактів додаються додаткові вузли, що представляють собою індикатори. Ребра в даному випадку направляються від фактів до індикаторів. В якості індикаторів використовуються метрики вихідного коду, що описані у розділі 3. В загальному випадку, метрика не повинна обчислюватися автоматично, а може бути обрахована вручну, або представляти собою деякі експертні знання виражені кількісно.

На рис. 13 зображено результат співставлення метрик вихідного коду, що описані у розділі 3, до відповідних фактів.

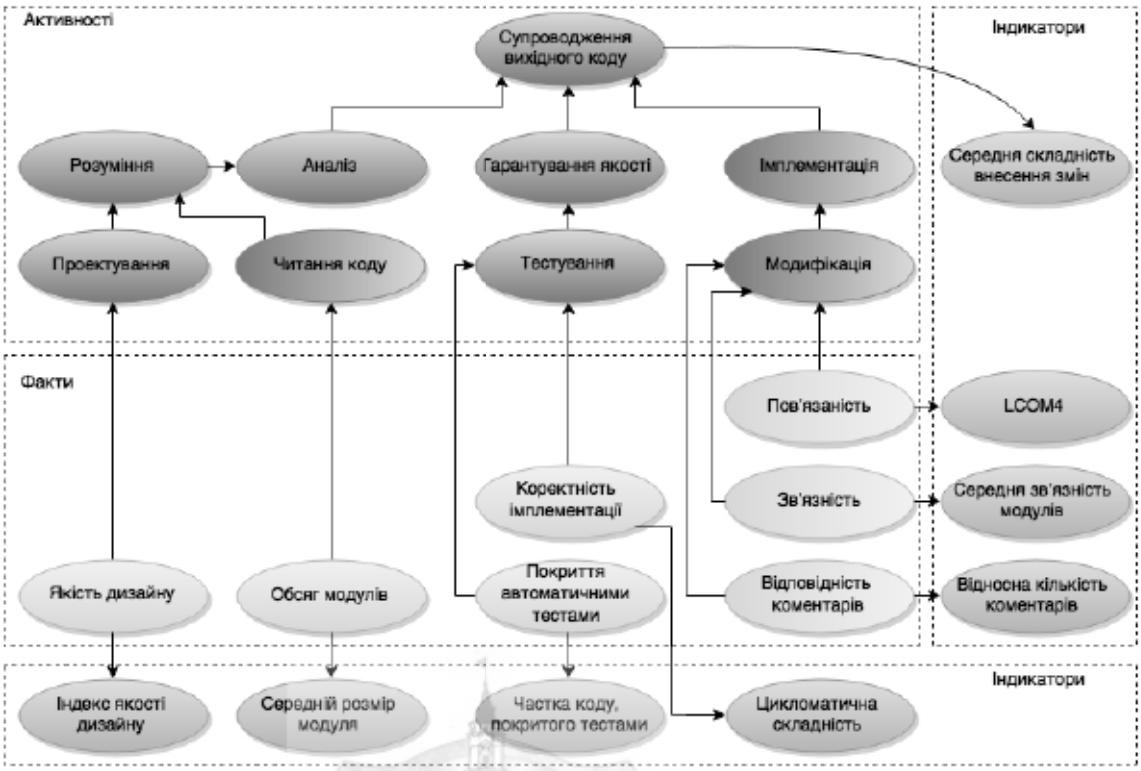


Рис. 13 – Результат визначення індикаторів відповідно до фактів

4.1.4. Заповнення таблиць ймовірностей

Для визначення таблиць ймовірностей вузлів використовується підхід, запропонований у [42]. Його основна ідея полягає у формалізації поведінки, що спостерігається у діях експертів, які повинні оцінити таблиці ймовірностей. Як правило, вони оцінюють центральну тенденцію або значення деяких екстремальних випадків, беручи за основу вершину, що здійснює вплив. Потім решта клітинок заповнюється відповідним чином. Такий підхід подібний до лінійної регресії, де для моделювання невизначеності використовується нормальний розподіл.

Стани вузлів, що являються індикаторами, залежать від масштабу метрики, яка використовується. Зазвичай, це дискретні стани або безперервні інтервали, наприклад, кількість рядків коду, що можуть вимірюватися в сотнях чи тисячах. Потім таблиці ймовірностей вузлів-індикаторів заповнюються або з використанням загальноприйнятих у індустрії значень, або інформацією на основі внутрішньо-корпоративних вимірювань.

4.2. Використання байесової мережі

Головною особливістю байесових мереж є їх здатність до моделювання різних сценаріїв. Маючи байесову мережу, побудовану за допомогою ABQM, можна ставити питання «Що як ...?». Ці питання формуються у сценарії, які можуть бути змодельовані та порівняні. Сценарій включає в себе додавання додаткової інформації до моделі, або, більш точно, додавання показання метрик до вузлів. В такому випадку, невизначеність зникає і наслідки для інших вузлів можуть бути пораховані. Байесова мережа дозволяє проводити виведення як у прямому, так і у зворотному напрямку, тобто, інформація може бути додана до будь-якого вузла, а ефект від цього обчислюється в усіх напрямках графу.

Найпростішим сценарієм використання є додавання виміряних значень метрик до вузлів-індикаторів. Це спричинить розрахунок вузлів-активностей. Вузол-активність у такому випадку покаже розподіл ймовірності для свого значення, тобто значення активності. Згодом, можуть бути внесені зміни у вузли-індикатори для моделювання інших сценаріїв. Це дозволить відобразити можливі зміни індикаторів на значення активностей для передбачення їх залежності.

4.3. Процес оцінювання

Підхід, описаний у попередньому розділі, може бути використаний у різних контекстах для оцінки та прогнозування мети. В цьому розділі, представлена початкова оцінка, використовуючи дані, що знаходяться у вільному доступі. При оцінюванні використано підхід, що базується на виділенні підмножини побудованої моделі якості, для якої є наявні дані у вільному доступі. Даний приклад демонструє основні принципи підходу з використанням наборів даних. Таким чином аналізується, чи виправданим є використання запропонованого підходу у майже реальних умовах. Рівень вірності прогнозування з використанням даних прикладів може свідчити про корисність підходу, а очікуваним значенням є, принаймні, отримання середніх результатів по індустрії. Вимірювання цих показників всередині компанії-розробника ПЗ потребуватиме додаткового часу та зусиль. Тільки тоді можливим є прийнятний аналіз з передбачення вірності моделі та порівняння з іншими моделями прогнозування.

Для моделювання байесової мережі використовується dlib [44] бібліотека. Вона може використовуватися для вирішення широкого кола задач, зокрема графічного моделювання, окремим випадком яких є байесові мережі. Бібліотека знаходить у вільному доступі та є зручною у використанні. Вона надає повний набір інструментів для моделювання байесових мереж, включаючи чисельні методи.

У роботі [45] наведені середні по індустрії значення необхідних зусиль для внесення змін у вихідний код. Середньою величино цього значення є 27,4 робочих години, де мінімумом є 3,9 години, а максимум – 66,6. Хоча внесення

змін не обов'язково означає виправлення помилок, це є досить точним для оцінювання побудованої моделі. Оскільки байесова мережа дає лише ймовірнісну оцінку того, що ПЗ є тої чи іншої якості, тому дані цифри використовуються як орієнтир для складання пропорції, враховуючи кількісне значення ймовірності стану, що є найбільш ймовірним.

4.4. Результати моделювання

Для проведення експерименту було обрано 4 системи з онлайн-ресурсу SonarQube [46]. SonarQube представляє собою відкриту платформу для безперервної інспекції якості коду. На цьому сайті міститься інформація про поточний стан великої кількості проектів (більше 200), що знаходяться у відкритому доступі. Інформація завжди є актуальною, бо оновлюється щодня. Тут зібрані проекти, написані з використанням різних мов програмування та з різних прикладних областей. Вихідний код кожного проекту піддається детальному аналізу, а результат складається з великої кількості метрик. Дані колекціонуються, та можливий перегляд стану проекту, починаючи з моменту його додавання на сайт. Також є доступною загальна статистика, підсумована по усім проектам, що можна вважати характеристикою індустрії загалом. Для вимірювання своєї оцінки, проект використовує:

- наявність та кількість дублікатів коду;
- порушення правил, що окремо встановлюються, виходячи з мови програмування, та їх вагу;
- рівень документування програмного інтерфейсу;
- покриття тестами;
- складність імплементації функцій та класів.

Детальний план оцінювання приведений на сайті онлайн-ресурсу [47].

У таблиці 4 приведені значення метрик для обраних проектів, виміряні за допомогою SonarQube станом на 15 червня 2015 р.

Таблиця 4 – Вихідні дані для моделювання

Метрика вихідного коду	Apache	Activity	ActiveMQ	SonarQube
Середній розмір модуля, к-сть рядків коду	91,67	58,7	95,0	57,7
Частка коду, покритого тестами, %	36,1	56,8	16,8	85,1
Цикломатична складність	2,7	1,9	2,7	1,9
Відносна кількість коментарів, %	21,6	13,8	15,1	7,6

На рис. 14 зображено розроблений програмний засіб для оцінки якості ПЗ. На даному рисунку описано загальну ситуацію, при якій жодних вимірювань не було зроблено. У полі результатів видно, що всі варіанти є рівно ймовірніснимми. На рис. 15 приведено результат роботи програмного засобу для проекту Doxygen.

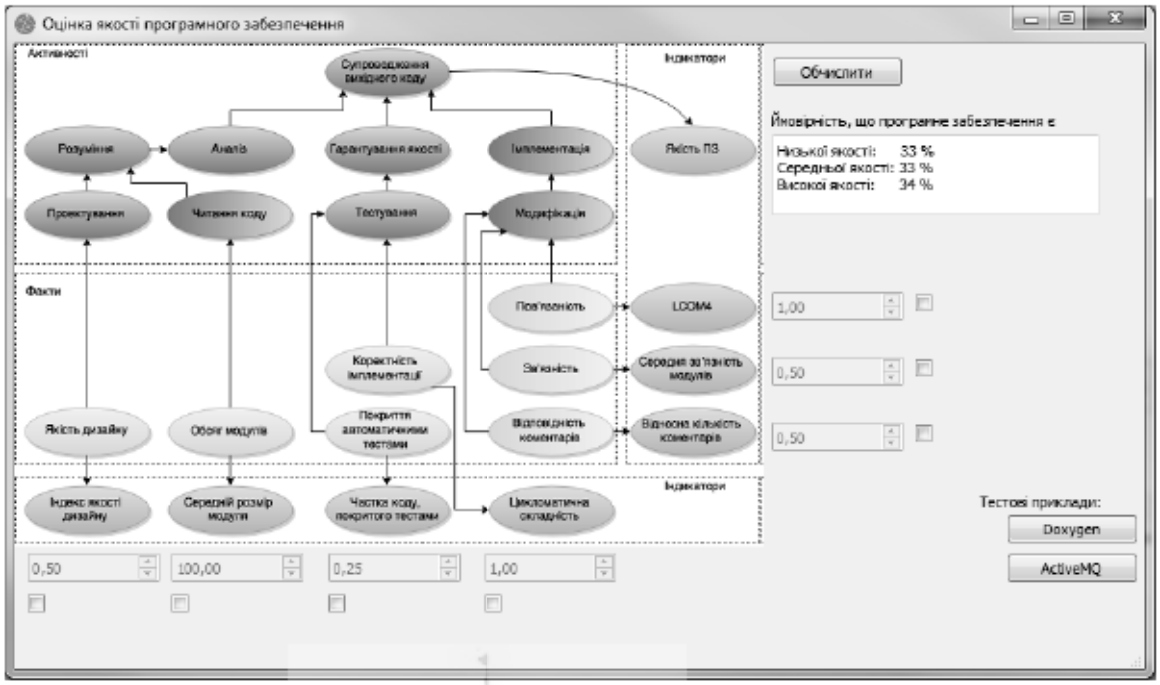


Рис. 14 – Розроблений програмний засіб для оцінки якості програмного забезпечення

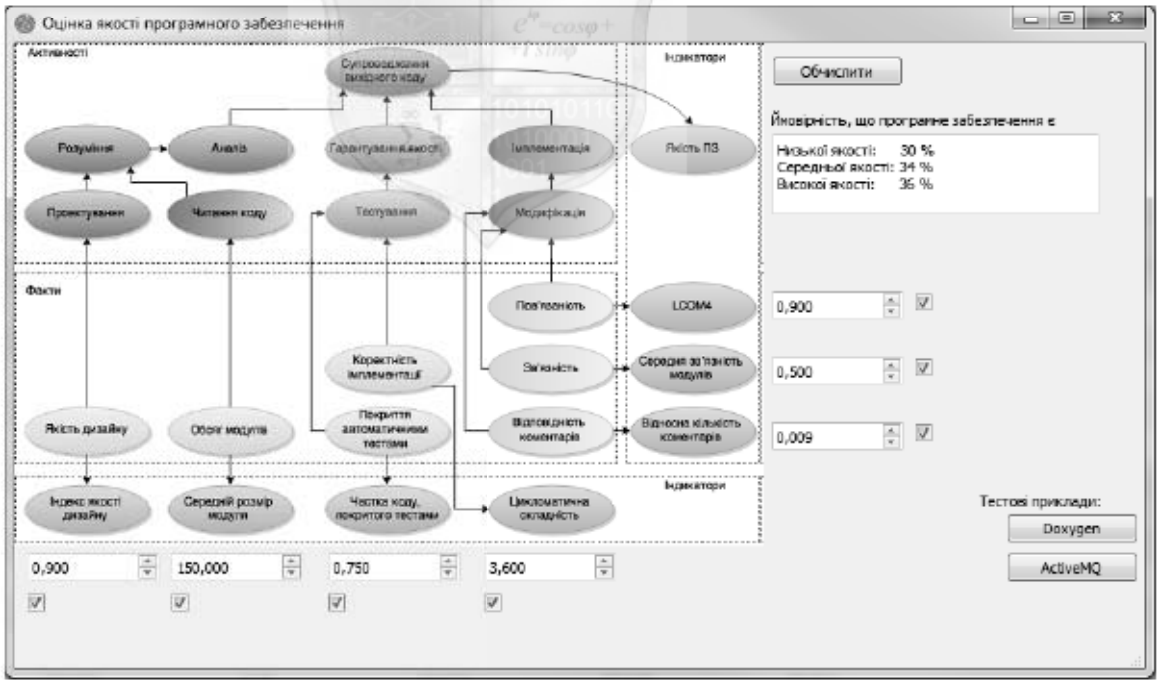


Рис. 15 – Приклад результату роботи програмного засобу

У таблиці 5 приведені результати моделювання. Також у таблиці приведено середній час усунення помилки, що прогнозує на основі своїх вимірювань SonarQube, та результат, що прогнозує модель, побудована у даній дисертації.

Таблиця 5 – Результати моделювання

Метрика вихідного коду	Apache	Activity	ActiveMQ	SonarQube
Прогнозоване значення SonarQube, години	44,9	47,5	39,0	22,0
Результати моделювання, години	40,72	41,9	40,5	19,57

Висновки

В даному розділі наведено приклад побудови та використання байесової мережі для оцінки якості програмного забезпечення. Створення байесової мережі відбувалося на основі моделі якості, що базується на активностях, з використанням GQM. Розробка програмних засобів, що реалізовують математичну модель, відбувалася з використанням новітніх інструментів розробки ПЗ. Так, для графічного моделювання байесової мережі було використано бібліотеку dlib, а графічний інтерфейс створено з залученням Qt5

[48]. Програмні засоби оцінки якості ПЗ реалізовані на мові програмування C++ та знаходяться у відкритому доступі [49], а також наведені у додатку Г.

В даному розділі сформульовані актуальні проблеми моделювання при використанні байесових мереж. Зокрема серед них виділяють проблему визначення таблиці ймовірностей для вузлів мережі. Проблема полягає у тому, що зі збільшенням кількості вузлів та станів кожного вузла, кількість даних у таблиці, що потребують кількісного вираження, що представляє собою співвідношення та взаємозв'язок між окремими вузлами та їх станами, зростає експоненційно. Перспективним напрямок розв'язання цієї проблеми є використання вузлів з нескінченною кількістю станів. В такому випадку, таблиця ймовірності заміняється функцією розподілу, яка визначає ймовірність того, що вузол перебуває у заданому стані (проміжку з функції розподілу). Проте, такий підхід потребує використання чисельних методів при моделюванні, що потребує в десятки разів більше ресурсів. У додатку Б наведено часова діаграма оцінювання якості ПЗ аналітичним та чисельним методом. Для прикладу використовувалася побудована у даній дисертації мережа. У якості чисельного методу для виведення байесової мережі було використано семплування за Гіббсом (алгоритм для генерації вибірки спільного розподілу множини випадкових величин), що складалося з 2000 ітерації. Час виконання семплування за Гіббсом перевищує час роботи аналітичного алгоритму майже у десять разів для одного і того ж самого випадку мережі. Результат роботи чисельного методу отримано з точністю до відсотка.

Порівняльний графік результатів оцінки якості ПЗ представлений у додатку В. По осі OX на графіку представлено результати оцінки, отримані за допомогою моделі, що побудована у даній дисертації. По осі OY відкладено прогнозоване значення відповідно до даних SonarQube. Чим ближче точка знаходиться до діагоналі, тим більш схожими є обидва результати.

ВИСНОВКИ

Однією з найголовніших цілей у галузі управління якістю ПЗ є надійне кількісне оцінювання та передбачення якості ПЗ. Зроблено багато продуктивних та корисних зусиль не тільки щодо побудови моделей оцінювання та прогнозування, а й в області визначення обмежень у використанні таких моделей. Тим не менше, ці моделі ще не є тісно інтегрованими у інші діяльності з управління якістю. Модель якості, що базується на активностях, на практиці довела, що може слугувати міцним фундаментом у задачах визначення якості на детальному рівні. Проте, використання кількісного аналізу було обмеженим.

На основі побудованої математичної моделі показано, що байесові мережі забезпечують хороші результати при прогнозуванні якості. Також вони мають ясну структуру, яка може бути дуже просто отримана, шляхом відображення моделі якості, що основана на активностях. Для виконання такого відображення, у даній роботі використано чотири-кроковий алгоритм трансформації. Це дозволяє систематично будувати байесові мережі, які використовують знання, що закодовані у моделі якості, для надання інформації щодо даного оцінювання чи прогнозування цілі. Було визначено модель якості (ABQM), яка у подальшому була збагачена оцінюванням якості та моделлю прогнозування якості.

За допомогою побудованої у даній роботі байесової мережі, була змодельована якість програмного забезпечення, що знаходиться у відкритому доступі. Список ПЗ для моделювання наведений у таблиці 4. Вихідні дані метрик було взято з онлайн-ресурсу SonarQube, де інформація оновлюється щодня, тому завжди є актуальною, а її зміни можна відслідковувати у часі. Додатковими сервісами цього онлайн-ресурсу також проводиться і аналіз

вихідного коду, на основі власних алгоритмів SonarQube, та оцінюється його якість. З цими показниками якості і було порівняно результати моделювання запропонованої у даній роботі байесової мережі. Результати моделювання представлені у таблиці 5 та у додатку В, де по осі OX на графіку представлено результати оцінки, отримані за допомогою моделі, що побудована у даній дисертації. По осі OY відкладено прогнозоване значення відповідно до даних SonarQube. З порівняльного графіку результатів видно, що оцінки є близькими одна до одної.

У додатку Б наведена порівняльна діаграма тривалості роботи розроблених програмних засобів з оцінювання якості ПЗ за допомогою байесової мережі. На діаграмі представлено необхідний час для отримання результату з використанням аналітичного рішення та чисельного методу. У якості чисельного методу для виведення байесової мережі було використано алгоритм для генерації вибірки спільного розподілу множини випадкових величин, що складається з 2000 ітерації. Час виконання чисельного методу перевищує час роботи аналітичного алгоритму майже у десять разів для одного і того ж самого випадку мережі та набору даних. Результат роботи чисельного методу отримано з точністю до відсотка.

У розділі 4 сформульовано актуальні проблеми моделювання при використанні байесових мереж. Зокрема серед них виділяють проблему визначення таблиці ймовірностей для вузлів мережі. Суть проблеми полягає у тому, що зі збільшенням кількості вузлів та станів кожного вузла, кількість даних у таблиці, що потребують кількісного вираження, що представляє собою співвідношення та взаємозв'язок між окремими вузлами та їх станами, зростає експоненційно. Це означає, що у випадку більш складних мереж, використання чисельного методу потребуватиме більше ітерацій семплування для досягнення тієї ж точності, а це, в свою чергу, збільшить час його роботи.

При розробці програмних засобів, що реалізовують математичну модель, було використано новітні програмні інструменти. Так, для графічного моделювання байесової мережі використано бібліотеку dlib. Графічний інтерфейс користувача побудовано за допомогою Qt5. Розробка виконана на мові програмування C++.

Використання байесових мереж відкриває багато можливостей. Після побудови байесової мережі великого розміру можна проводити аналіз чутливості вхідних параметрів цієї мережі на результат. Це може дати відповідь на дуже важливі на практиці питання щодо того, які параметри системи є найважливішими, і, відповідно, суттєво зменшити необхідні зусилля на вимірювання неважливих компонентів.

Подальші дослідження варто спрямувати на вивчення узагальнених випадків байесової мережі, що складаються з вузлів, які описуються неперервними функціями, а, отже, мають нескінченну кількість станів. Замість таблиці ймовірності в такому випадку повинна задаватися функція розподілу. Це потребуватиме також дослідження чисельних методів перенесення вірогідності того чи іншого показання метрики через мережу.

Окремим перспективним напрямом досліджень вважається вивчення та розробка алгоритмів навчання байесових мереж з метою поліпшення процесу визначення таблиць імовірності, можливо – на основі емпіричних даних. Це стосується як дискретних, так і неперервних вузлів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Українська IT індустрія – 20 років розвитку [Електронний ресурс] // Miratech. – 2012. – Режим доступу : http://miratech.ua/sites/default/files/images/acc_ict_miratech_ukr.pdf. – Дата доступу : червень 2015. – Назва з екрана.
2. Yourdon, E. Death March [Text] / Edward Yourdon. – 2nd edition. – USA: Prentice Hall, 2003. – 256 p.
3. Brooks, F. The Mythical Man-Month: Essays on Software Engineering [Text] / Frederick P. Brooks Jr. – 2nd edition. – USA: Addison-Wesley Professional, 1995. – 336 p.
4. Fenton, Norman E.; Neil, Martin. Software metrics: roadmap. Proceedings of the Conference on the Future of Software Engineering . ACM, 2000. p. 357-370.
5. Чертов О.Р., Довгаль К.І. Способи розробки моделі оцінки якості програмного забезпечення // Прикладна математика та комп'ютинг. ПМК, 2015 : сьома наук. конф. магістрантів та аспірантів, Київ, 15—17 квіт. 2015 р. : зб. тез доп. / [редкол.: Дичка І. А. та ін.]. — К. : Просвіта, 2015.
6. Rousseeuw, P. J.; Leroy, A. M. Robust regression and outlier detection [Text] / Rousseeuw, Peter J.; Leroy, Annick M John. – Ney Work. – Wiley & Sons, 2005. – 360 p.
7. Gray, Andrew R.; MacDonnell, Stephen G. A comparison of techniques for developing predictive models of software metrics. Information and software technology, 1997, 39.6: 425-437.

8. Hettmansperger, Thomas P.; Sheather, Simon J. A cautionary note on the method of least median squares. The American Statistician, 1992, 46.2: 79-83.
9. Fedotova, Olga; Teixeira, Leonor; Alvelos, Helena. Software Effort Estimation with Multiple Linear Regression: Review and Practical Application. Journal of Information Science and Engineering, 2013, 29.5: 925-945.
10. Briand, Lionel C., et al. An assessment and comparison of common software cost estimation modeling techniques. Proceedings of the 21st international conference on Software engineering. ACM, 1999. p. 313-322.
11. Finnie, Gavin R.; Wittig, Gerhard E.; Desharnais, Jean-Marc. A comparison of software effort estimation techniques: using function points with neural networks, case-based reasoning and regression models. Journal of Systems and Software, 1997, 39.3: 281-289.
12. Li, Eldon Y. Artificial neural networks and their business applications. Information & Management, 1994, 27.5: 303-313.
13. Wittig, Gerhard; Finnie, Gavin. Estimating software development effort with connectionist models. Information and Software Technology, 1997, 39.7: 469-476.
14. WANG, Li-Xin; MENDEL, Jerry M. Generating fuzzy rules by learning from examples. Systems, Man and Cybernetics, IEEE Transactions on, 1992, 22.6: 1414-1427.
15. Kumar, Satish; Krishna, B. Ananda; Satsangi, Prem S. Fuzzy systems and neural networks in software engineering project management. Applied Intelligence, 1994, 4.1: 31-52.
16. Bastani, Farokh B.; DiMarco, Giuseppe; Pasquini, Alberto. Experimental evaluation of a fuzzy-set based measure of software correctness using program mutation. In: Proceedings of the 15th international conference on

Software Engineering. IEEE Computer Society Press, 1993. p. 45-54.

17. Munakata, Toshinori; Jani, Yashvant. Fuzzy systems: an overview. Communications of the ACM, 1994, 37.3: 68-76.
18. Kosko, Bart. Fuzzy systems as universal approximators. Computers, IEEE Transactions on, 1994, 43.11: 1329-1333.
19. Castro, Juan Luis. Fuzzy logic controllers are universal approximators. Systems, Man and Cybernetics, IEEE Transactions on, 1995, 25.4: 629-635.
20. Ramsey, Connie Loggia; Basili, Victor R. An evaluation of expert systems for software engineering management. Software Engineering, IEEE Transactions on, 1989, 15.6: 747-759.
21. Lakhotia, Arun. Rule-based approach to computing module cohesion. In: Proceedings of the 15th international conference on Software Engineering. IEEE Computer Society Press, 1993. p. 35-44.
22. Wagner, Stefan. A Bayesian network approach to assess and predict software quality using activity-based quality models. Information and Software Technology, 2010, 52.11: 1230-1241.
23. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, et al. Software engineering – Product quality – Part 1: Quality model. ISO/IEC, 2001, 9126: 2001.
24. Deissenboeck, Florian, et al. An activity-based quality model for maintainability. In: Software Maintenance, 2007. ICSM 2007. IEEE International Conference on. IEEE, 2007. p. 184-193.
25. INTERNATIONAL ORGANIZATION FOR STANDARDIZATION / INTERNATIONAL ELECTROTECHNICAL COMMISSION, et al. Software Engineering – Software Life Cycle Processes – Maintenance. ISO/IEC, 2006, 14764:2006.

26. Довгаль К.І., Чертов О.Р. Байесові мережі для моделей оцінки якості програмного забезпечення // Системний аналіз та інформаційні технології: матеріали 17-ї Міжнародної науково-технічної конференції SAIT 2015, Київ, 22-26 червня 2015 р. / ННК «ІПСА» НТУУ «КПІ». – К.: ННК «ІПСА» НТУУ «КПІ», 2015.
27. Fenton, Norman E.; Neil, Martin. Software metrics: roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM, 2000. p. 357-370.
28. Chidamber, Shyam R.; Kemerer, Chris F. A metrics suite for object oriented design. Software Engineering, IEEE Transactions on, 1994, 20.6: 476-493.
29. Basili, Victor R.; Reiter JR, Robert W. Evaluating automatable measures of software development. In: Proceedings on Workshop on Quantitative Software Models. 1979. p. 107-116.
30. Weyuker, Elaine J. Evaluating software complexity measures. Software Engineering, IEEE Transactions on, 1988, 14.9: 1357-1365.
31. DSQI (Design Structure Quality Index) [Електронний ресурс] // Code Buzz. – 2012. – Режим доступу : http://logicalprogram.blogspot.com/p/dsqi.html. – Дата доступу : червень 2015. – Назва з екрана.
32. Design Structure Quality Index (DSQI) Calculator [Електронний ресурс] // Tiny tools. – 2012. – Режим доступу : http://groups.engin.umd.umich.edu/CIS/tinytools/cis375/f00/dsqi/main.html. – Дата доступу : червень 2015. – Назва з екрана.
33. Fenton, Norman E.; Neil, Martin. Software metrics: successes, failures and new directions. Journal of Systems and Software, 1999, 47.2: 149-157.
34. Miller, Joan C.; Maloney, Clifford J. Systematic mistake analysis of digital computer programs. Communications of the ACM, 1963, 6.2: 58-63.
35. McCabe, Thomas J. A complexity measure. Software Engineering, IEEE Transactions on, 1976, 4: 308-320.

36. Hitz, Martin; Montazeri, Behzad. Measuring coupling and cohesion in object-oriented systems. na, 1995.
37. Yourdon, Edward; Constantine, Larry L. Structured design: Fundamentals of a discipline of computer program and systems design. Prentice-Hall, Inc., 1979.
38. Pressman, Roger S. Software engineering: a practitioner's approach. Palgrave Macmillan, 2005.
39. Arafat, Oliver; Riehle, Dirk. The comment density of open source software code. In: Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on. IEEE, 2009. p. 195-198.
40. Goodman, Paul. Software metrics: Best practices for successful IT management. Rothstein Associates Inc, 2004.
41. Fenton, Norman; Neil, Martin. Managing Risk in the Modern World. Application of Bayesian Networks, 2007.
42. Fenton, Norman E.; Neil, Martin; Caballero, Jose Galan. Using ranked nodes to model qualitative judgments in Bayesian networks. Knowledge and Data Engineering, IEEE Transactions on, 2007, 19.10: 1420-1432.
43. Fenton, Norman E.; Neil, Martin. A critique of software defect prediction models. Software Engineering, IEEE Transactions on, 1999, 25.5: 675-689.
44. Dlib C++ Library [Електронний ресурс] // Dlib. – 2015. – Режим доступу : http://dlib.net/. – Дата доступу : червень 2015. – Назва з екрана.
45. Wagner, Stefan. A literature survey of the quality economics of defect-detection techniques. In: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering. ACM, 2006. p. 194-203.
46. Put your technical debt under control [Електронний ресурс] // SonarQube. – 2015. – Режим доступу : http://www.sonarqube.org/. – Дата доступу : червень 2015. – Назва з екрана.

47. Evaluate your technical debt with Sonar [Електронний ресурс] // SonarQube. – 2009. – Режим доступу : http://www.sonarqube.org/evaluate-your-technical-debt-with-sonar/. – Дата доступу : червень 2015. – Назва з екрана.
48. Qt Application Development [Електронний ресурс] // Qt. – 2009. – Режим доступу : http://www.qt.io/application-development/. – Дата доступу : червень 2015. – Назва з екрана.
49. Assess quality of source code using Bayesian network [Електронний ресурс] // GitHub. – 2015. – Режим доступу : https://github.com/CDovgal/software-quality-model/tree/dlib_branch. – Дата доступу : червень 2015. – Назва з екрана.