

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки комп'ютерної та програмної інженерії
Кафедра інженерії програмного забезпечення



ЛАБОРАТОРНА РОБОТА №1.2
Застосування метрик і моделей якості
З дисципліни «Якість програмного забезпечення та тестування»

Роботу виконав студент
групи ПІ-3226
Бабій В.І.
Роботу перевірів викладач
Корнієнко С.П.

Мета:

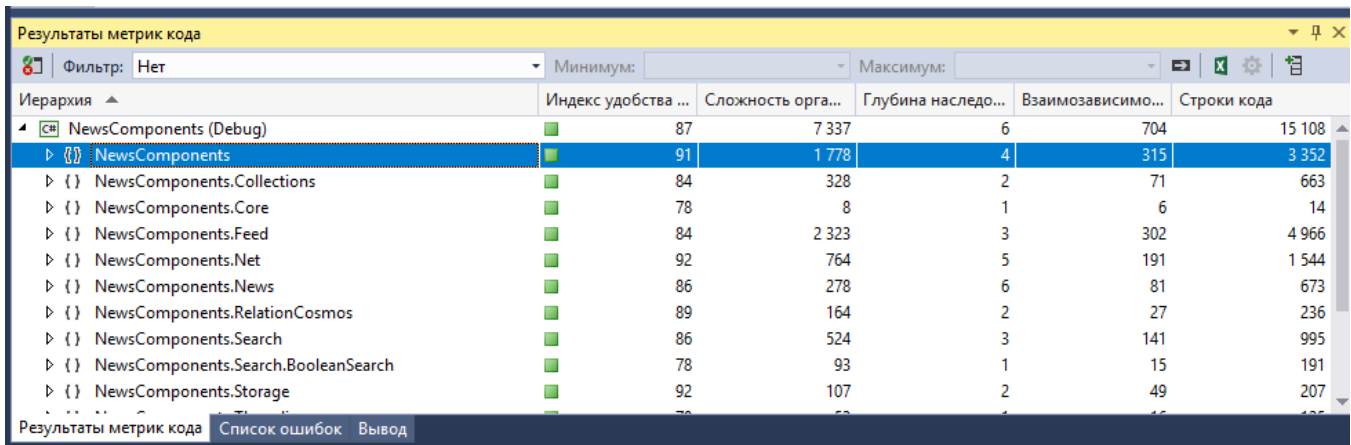
Навчитися використовувати засоби метрики коду VSTS для його аналізу.

Завдання:

- 1) Завантажити і встановити програму RSS Bandit application і її програмний код.
- 2) Відкрити код в Visual Studio 2020 (або іншій версії).
- 3) Згенерувати Метрику Коду для всього коду;
- 4) Проаналізувати результат; знайти “гнилий код” і спробувати зробити рефакторинг;
- 5) Написати звіт з метрикою коду і з покращеним кодом;
- 6) Вивчити теорію метрик коду для захисту лабораторної роботи.

Хід роботи

1. Згідно завдання, я взяв частину коду RSS Bandit
2. Проаналізувавши результат, було виявлено код, показники якого можна покращити.
3. Після аналізу коду було виконано рефакторинг та аналіз нових результатів метрик.
4. Результати метрик коду до рефакторингу та після показано на рисунках:



Результаты метрик кода

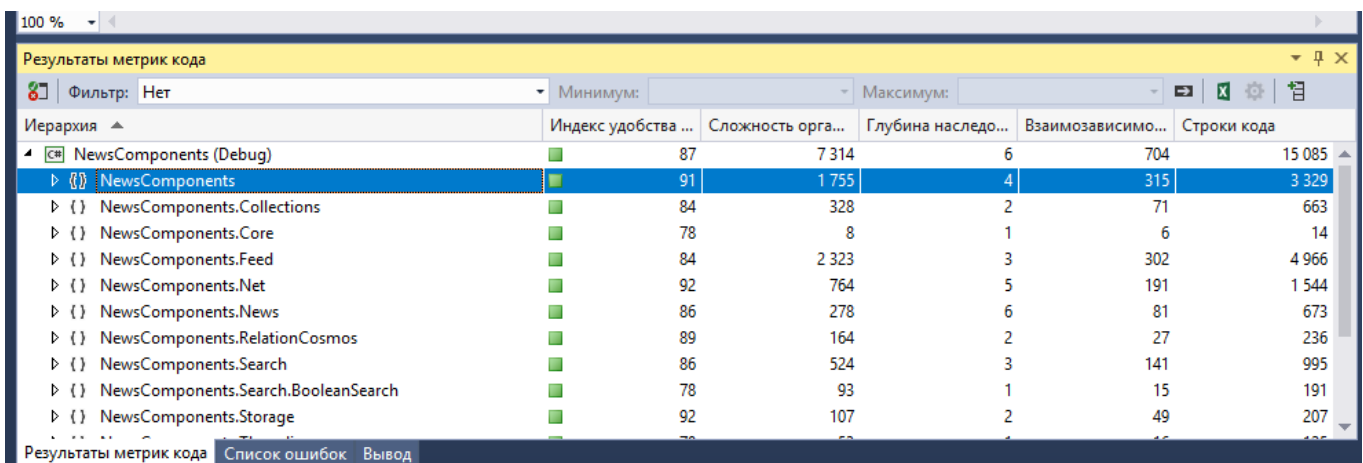
Фильтр: Нет

Минимум: Максимум:

Иерархия	Индекс удобства ...	Сложность орга...	Глубина наследо...	Взаимозависимо...	Строки кода
NewsComponents (Debug)	87	7 337	6	704	15 108
NewsComponents	91	1 778	4	315	3 352
NewsComponents.Collections	84	328	2	71	663
NewsComponents.Core	78	8	1	6	14
NewsComponents.Feed	84	2 323	3	302	4 966
NewsComponents.Net	92	764	5	191	1 544
NewsComponents.News	86	278	6	81	673
NewsComponents.RelationCosmos	89	164	2	27	236
NewsComponents.Search	86	524	3	141	995
NewsComponents.Search.BooleanSearch	78	93	1	15	191
NewsComponents.Storage	92	107	2	49	207

Результаты метрик кода | Список ошибок | Вывод

Метрики коду до рефакторингу



100 %

Результаты метрик кода

Фильтр: Нет

Минимум: Максимум:

Иерархия	Индекс удобства ...	Сложность орга...	Глубина наследо...	Взаимозависимо...	Строки кода
NewsComponents (Debug)	87	7 314	6	704	15 085
NewsComponents	91	1 755	4	315	3 329
NewsComponents.Collections	84	328	2	71	663
NewsComponents.Core	78	8	1	6	14
NewsComponents.Feed	84	2 323	3	302	4 966
NewsComponents.Net	92	764	5	191	1 544
NewsComponents.News	86	278	6	81	673
NewsComponents.RelationCosmos	89	164	2	27	236
NewsComponents.Search	86	524	3	141	995
NewsComponents.Search.BooleanSearch	78	93	1	15	191
NewsComponents.Storage	92	107	2	49	207

Результаты метрик кода | Список ошибок | Вывод

Метрики коду після рефакторингу

Висновок: в ході лабораторної роботи було зроблено аналіз метрик коду до та після рефакторингу, після якого метрики коду незначним чином покращились.

Відповіді на контрольні питання.

Кількісна оцінка якості

Що таке хороша програма і чим вона відрізняється від поганої? Сьогодні таке питання зазвичай трактується в плані оцінки якості програми кінцевим користувачем. Але насправді він актуальний і для самих розробників ПЗ, правда, при цьому вони в першу чергу мають на увазі не споживчі властивості програми, а якість її коду. Власне, саме в такій постановці питання про "хорошу програму" було поставлене ще в кінці 60-х років, коли з'явилися перші дослідження в цій області.

Зрозуміло, що "поганий /хороший код" оцінюється на етапах розробки і супроводу. У цій ситуації якість коду визначається такими показниками, як правильне розбиття програми на модулі, обмеження у використанні потенційно ризикованих мовних конструкцій, наочне оформлення вихідного коду.

Але навіть визначивши склад ключових показників (метрик) якості коду, практично неможливо створити універсальні критерії, що дозволяють вважати програму "поганою" або "хорошою". У будь-якому випадку така оцінка носить дуже суб'єктивний характер, і навіть для одного програміста вона буде варіюватися залежно від типу проекту. Тому існуючі інструменти визначення метрик коду (code metrics, CM) в основному обмежуються обчисленням відповідних значень, інтерпретація яких повністю покладається на людину.

Раніше для виконання подібних завдань у середовищі Microsoft Visual Studio потрібно було використовувати додаткові кошти третіх фірм. Але тепер на ринку ПЗ з'явилася Visual Studio (від 2008) Team Edition for the Software Developer.

Ось за якими метриками коду можна стежити вже зараз:

- індекс експлуатаційної надійності (Maintainability Index, MI) - комплексний показник якості коду (від 0 до 100 - чим вище, тим краще); методика його визначення розроблена фахівцями Carnegie Mellon Software Engineering Institute;
- циклічна складність (Cyclomatic Complexity, CC) - показник, що характеризує число гілок в програмному коді та обчислюється шляхом підрахунку операторів циклу, умовного переходу;
- глибина успадкування (Depth of Inheritance) - характеризує довжину ланцюжків спадкування в програмному коді;
- зчеплення класів (Class Coupling) – відображає ступінь залежності класів між собою (в тому числі наявність спільних даних, об'єктів тощо);
- число рядків коду: чим більше рядків, тим складніше програма.

Звичайно, для більш детального аналізу якості коду бажано використовувати більше показників, які можна визначати за допомогою засобів третіх фірм (є чимало і безкоштовних продуктів). Але, як правило, більш широкий спектр характеристик потрібно тільки в навчальних цілях і не застосовується розробниками на практиці.

2.1. Метрики якості

Метрика програмного забезпечення (англ. software metric) — це міра, яка дозволяє отримати числове значення деякої властивості ПЗ.

У загальному випадку застосування метрик дозволяє керівникам проектів вивчити складність розробленого проекту, оцінити обсяг робіт і зусилля, витрачені кожним розробником для реалізації того чи іншого рішення.

Однак метрики можуть служити лише рекомендаційними характеристиками, ними не можна повністю керуватися, тому що при розробці ПЗ програмісти, прагнучи мінімізувати або максимізувати ту чи іншу міру для своєї програми, можуть вдаватися до хитрощів аж до зниження ефективності

роботи програми. Крім того, якщо, наприклад, програміст написав малу кількість рядків коду або вніс невелике число структурних змін, це зовсім не означає, що він нічого не робив, а може означати, що дефект програми було дуже складно відшукати. Остання проблема, однак, частково може бути вирішена при використанні метрик складності, тому що в більш складній програмі помилку знайти складніше.

Метрики складності програм прийнято поділяти на три основні групи:

- метрики розміру програм;
- метрики складності потоку управління програм;
- метрики складності потоку даних програм.

Метрики першої групи базуються на визначенні кількісних характеристик, пов'язаних з розміром програми, і відрізняються відносною простотою. До найбільш відомих метрика цієї групи відносяться кількість операторів програми, кількість рядків вихідного тексту. Метрики цієї групи орієнтовані на аналіз вихідного тексту програм. Тому вони можуть використовуватися для оцінки складності проміжних продуктів розробки.

Метрики другої групи базуються на аналізі керуючого графа програми. Представником цієї групи є метрика Маккейба. Керуючий граф (блок-схема програми), який використовують метрики даної групи, може бути побудований на основі алгоритмів модулів. Тому метрики другої групи можуть застосовуватися для оцінки складності проміжних продуктів розробки.

Метрики третьої групи базуються на оцінці використання, конфігурації і розміщення даних у програмі. У першу чергу це стосується глобальних змінних. До цієї групи відносяться метрики Чепіна.

Розмірно-орієнтовані метрики

2.1.1. Лос- оцінки якості

Розмірно-орієнтовані метрики прямо вимірюють програмний продукт. Грунтуються такі метрики на LOC-оцінках.

Кількість рядків вихідного коду (Lines of Code - LOC) є найбільш простим і розповсюдженим способом оцінки обсягу робіт за проектом.

Спочатку даний показник виник як спосіб оцінки обсягу роботи за проектом, в якому застосовувалися мови програмування, що володіють досить простою структурою: «один рядок коду = одна команда мови». Також давно відомо, що одну й ту ж функціональність можна написати різною кількістю рядків, а якщо візьмемо мову високого рівня (C ++, Java), то можливо і в одному рядку написати функціонал 5-6 рядків – це не проблема.

Тому метод LOC є тільки оціночним методом (який треба брати до відома, але не спиратися в оцінках) і ніяк не обов'язковим. Залежно від того, яким чином враховується подібний код, виділяють два основні показники SLOC:

- кількість «фізичних» рядків коду визначається як загальне число рядків вихідного коду, включаючи коментарі і порожні рядки (при вимірюванні показника на кількість порожніх рядків, як правило, вводиться обмеження -- при підрахунку враховується кількість порожніх рядків, що не перевищує 25% загального числа рядків у вимірюваному блоці коду).

- кількість «логічних» рядків коду (використовуються аббревіатури LSI, DSI, KDSI, де «SI» - source instructions) - визначається як кількість команд і залежить від мови програмування. У тому випадку, якщо мова не допускає розміщення кількох команд в одному рядку, то кількість «логічних» буде відповідати числу «фізичних», за винятком числа порожніх рядків і рядків коментарів. У тому випадку, якщо мова програмування підтримує розміщення кількох команд в одному рядку, то один

фізичний рядок повинен бути врахована як кілька логічних, якщо вона містить більше однієї команди мови. Цей варіант має також свої недоліки, так як залежить від мови програмування та стилю.

Для метрики SLOC існує велике число похідних, покликаних отримати окремі показники проекту, основними серед яких є:

- кількість порожніх рядків;
- число рядків, що містять коментарі;
- відсоток коментарів (відношення рядків коду до рядків коментаря, похідна метрика стилістики);
- середнє число строк для функцій (класів, файлів);
- середнє число рядків, що містять вихідний код для функцій (класів, файлів);
- середнє число строк для модулів.

Недоліки SLOC

Потенційні недоліки SLOC, на які орієнтована критика:

- неправильно зводити оцінку роботи людини до декількох числових параметрів і по них судити про продуктивність. Менеджер може призначити найбільш талановитих програмістів на складний ділянку роботи; це означає, що розробка цієї ділянки займе найбільший час і породить найбільшу кількість помилок, через складність завдання. Не знаючи про ці труднощі, інший менеджер з отриманими показниками може вирішити, що програміст зробив свою роботу погано.

- Спотворення: процес вимірювання може бути спотворений за рахунок того, що співробітники знають про вимірюваних показниках і прагнуть оптимізувати ці показники, а не свою роботу. Наприклад, якщо кількість рядків вихідного коду є важливим показником, то програмісти будуть прагнути писати якомога більше рядків і не будуть використовувати способи спрощення коду, що скорочують кількість рядків.

- Неточність: ні метрик, які були б одночасно і значущі і досить точні. Кількість рядків коду - це просто кількість рядків, цей показник не дає уявлення про складність вирішуваної проблеми. Аналіз функціональних точок був розроблений з метою кращого вимірювання складності коду і специфікації, але він використовує особисті оцінки вимірювального, тому різні люди отримують різні результати.

І головне пам'ятати: метрика SLOC не відображає трудомісткості по створенню програми.

В організаціях, зайнятих розробкою програмної продукції для кожного проекту прийнято реєструвати наступні показники:

- загальні трудовитрати (в людино-місяцях, людино-годинах);
- обсяг програми (в тисячах рядках вихідного коду-LOC);
- вартість розробки;
- обсяг документації;
- помилки, виявлені протягом року експлуатації;
- кількість людей, які працювали над виробом;
- термін розробки.

LOC суттєво залежить від мови програмування.

Приклад з життя [26]:

На наш погляд оцінка за кількістю рядків у коді тягне за собою спокусу написати більше рядків, щоб взяти побільше грошей. Зрозуміло, про оптимізацію в такому продукті ніхто вже думати не стане. Згадаймо історію про те, як планетарний центр аутсорсингу - Індія, після того, як замовники поставив їм метрику LOC, на другий день показав подвоєння і потроєння рядків коду.

Приклад з життя [26]:

В одній з компаній при впровадженні ми застосували цю метрику. Керівник організації був у відпустці, але після повернення з неї вирішив скористатися прозорістю та трасуванню змін і подивитися, як же йдуть справи в проектах у його менеджерів. І щоб повністю увійти в курс, опустився на найнижчий рівень (тобто не став оцінювати щільність дефектів, кількість виправлених багів) - на рівень вихідних текстів. Вирішив порахувати, хто і скільки рядків написав. А щоб було зовсім весело - співвіднести кількість робочих днів на тиждень і кількість написаного коду (логіка проста: людина працювала 40 годин на тиждень, отже, має багато чого написати). Природно, знайшлася людина, яка за тиждень написав всього один рядок, навіть не написав, а тільки відкоригував існуючу ...Гніву керівника не було меж - знайшов нероби! І погано було б програмісту, якби менеджер проекту не пояснив, що: була знайдена помилка у програмі, знайшов її VIP-клієнт, помилка впливає на бізнес клієнта і її треба було терміново усунути, для цього був обраний ось цей конкретний виконавець, який розгорнув стенд, залив середу клієнта, підтвердив прояв помилки і почав її шукати і усувати. Природно, врешті-решт, він поміняв фрагмент коду, в якому було неправильне умова і все запрацювало. Погодьтеся, вважати трудовитрати з даної метриці нерозумно - необхідна комплексна оцінка ...

2.1.2. Метрики стилістики й зрозумілості програм

Іноді важливо не просто порахувати кількість рядків коментарів в коді і просто співвіднести з логічними рядками коду, а дізнатися щільність коментарів. Тобто код спочатку був документований добре, потім - погано. Або такий варіант: шапка функції або класу документована і прокоментована, а код немає коментарів.

$$Fi = \text{SIGN} (N_{\text{комм. } i} / Ni - 0,1),$$

Де $N_{\text{комм. } i}$ - кількість коментарів в i -му фрагменті

Ni - загальна кількість рядків коду в i -му фрагменті

Суть метрики проста: код розбивається на n -рівних частин і для кожного з них визначається Fi
Загальна оцінка визначається як сума $F = \sum F_i$.

Метрики складності

Крім показників оцінки обсягу робіт за проектом дуже важливими для отримання об'єктивних оцінок за проектом є показники оцінки його складності. Як правило, дані показники не можуть бути обчислені на самих ранніх стадіях роботи над проектом, оскільки вимагають, як мінімум, детального проектування. Однак ці показники дуже важливі для отримання прогнозних оцінок тривалості і вартості проекту, оскільки безпосередньо визначають його трудомісткість.

2.1.3. Об'єктно-орієнтовані метрики

У сучасних умовах більшість програмних проектів створюється на основі ОО підходу, у зв'язку з чим існує значна кількість метрик, що дозволяють отримати оцінку складності об'єктно-орієнтованих проектів.

Зважена насиченість класу (Weighted Methods Per Class (WMC)) Відображає відносну міру складності класу на основі циклічної складності кожного його методу. Клас з більш складними методами і великою кількістю методів вважається більш складним. При обчисленні метрики батьківські класи не враховуються.

Глибина дерева успадкування (Depth of inheritance tree) Довжина найдовшого шляху спадкоємства. Чим глибше дерево успадкування модуля, тим може опинитися складніше передбачити його поведінку. З іншого боку, збільшення глибини дає більший потенціал повторного використання даним модулем поведінки, визначеного для класів- предків.

Кількість нащадків (Number of children) Число модулів, безпосередньо успадковують даний модуль. Більші значення цієї метрики вказують на широкі можливості повторного використання; при цьому занадто велике значення може свідчити про погано вибраної абстракції.

Зв'язність об'єктів (Coupling between objects) Кількість модулів, пов'язаних з даним модулем в ролі клієнта або постачальника. Надмірна зв'язність говорить про слабкість модульної інкапсуляції і може перешкоджати повторного використання коду.

Відгук на клас (Response For Class) Кількість методів, які можуть викликатися екземплярами класу; обчислюється як сума кількості локальних методів, так і кількості вилучених методів

2.1.4. Метрики Холстеда

Метрика Холстед відноситься до метрика, обчислюваних на підставі аналізу числа рядків і синтаксичних елементів початкового коду програми.

Основу метрики Холстеда складають **чотири вимірювані характеристики програми:**

- NUOprtr (Number of Unique Operators) - число унікальних операторів програми, включаючи символи-роздільники, імена процедур і знаки операцій (словник операторів);
- NUOprnd (Number of Unique Operands) - число унікальних операндів програми (словник операндів);
- Noprtr (Number of Operators) - загальна кількість операторів в програмі;
- Noprnd (Number of Operands) - загальна кількість операндів в програмі.

На підставі цих характеристик розраховуються оцінки:

- **Словник програми** (Halstead Program Vocabulary, HPVoc): $HPVoc = NUOprtr + NUOprnd$;
- **Довжина програми** (Halstead Program Length, HPLen): $HPLen = Noprtr + Noprnd$;
- **Обсяг програми** (Halstead Program Volume, HPVol): $HPVol = HPLen \log_2 HPVoc$;
- **Складність програми** (Halstead Difficulty, HDiff): $HDiff = (NUOprtr / 2) \times (Noprnd / NUOprnd)$;
- На основі показника HDiff пропонується оцінювати **зусилля програміста при розробці** за допомогою показника HEff (Halstead Effort): $HEff = HDiff \times HPVol$.

2.1.5. Метрики циклічної складності за Мак-Кейбом

Показник циклічної складності є **одним з найпоширеніших показників оцінки складності програмних проектів**. Даний показник був розроблений вченим Мак-Кейбом в 1976 р., належить до групи показників оцінки складності потоку управління програмою і обчислюється на основі графа керуючої логіки програми (*control flow graph*). Даний граф будується у вигляді орієнтованого графа, в якому обчислювальні оператори або вирази представляються у вигляді вузлів, а передача управління між вузлами - у вигляді дуг.

Показник циклічної складності дозволяє не тільки зробити оцінку трудомісткості реалізації окремих елементів програмного проекту і скорегувати загальні показники оцінки тривалості і вартості проекту, а й оцінити пов'язані ризики і прийняти необхідні управлінські рішення.

Спрощена формула обчислення циклічної складності представляється наступним чином:

$$C = e - n + 2,$$

де e - число ребер, а n - число вузлів на графі керуючої логіки.

Як правило, при обчисленні циклічної складності логічні оператори не враховуються.

У процесі автоматизованого обчислення показника циклічної складності, як правило, застосовується спрощений підхід, відповідно до якого побудова графа не здійснюється, а обчислення показника проводиться на підставі підрахунку кількості операторів керуючої логіки (if, switch і т.д.) і можливої кількості шляхів виконання програми.

Цикломатичне число Мак-Кейба показує необхідну кількість проходів для покриття всіх контурів сильно зв'язаного графу або кількості тестових прогонів програми, необхідних для вичерпного тестування за принципом «*працює кожна гілка*».

Показник циклічної складності може бути розрахований для модуля, методу та інших структурних одиниць програми.

Існує значна кількість модифікацій показника циклічної складності.

- «Модифікована» цикломатична складність - розглядає не кожне розгалуження оператора множинного вибору (switch), а весь оператор як єдине ціле.
- «Строго» цикломатична складність - включає логічні оператори.
- «Спрощена» обчислення циклічної складності – передбачає обчислення не на основі графа, а на основі підрахунку керуючих операторів.

2.1.6. Метрики Чепіна

Існує кілька її модифікацій. Розглянемо більш простий, а з точки зору практичного використання - досить ефективний варіант цієї метрики.

Суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля за допомогою аналізу характеру використання змінних зі списку вводу-виводу.

Всі безліч змінних, що складають список вводу-виводу, розбивається на чотири функціональні групи.

1. «Р» - змінні для розрахунків та для забезпечення виведення. Прикладом може служити що використовується в програмах лексичного аналізатора змінна, що містить рядок вихідного тексту програми, тобто сама мінлива не модифікується, а лише містить вихідну інформацію.
2. «М» - модифікуються або створювані усередині програми змінні.
3. «С» - змінні, що беруть участь в управлінні роботою програмного модуля (керуючі змінні).
4. «Т» - не використовуються в програмі ("паразитні") змінні. Оскільки кожна змінна може виконувати одночасно декілька функцій, необхідно враховувати її в кожній відповідній функціональній групі.

Далі вводиться значення метрики Чепіна:

$$Q = a_1P + a_2M + a_3C + a_4T,$$

де a_1 , a_2 , a_3 , a_4 - вагові коефіцієнти.

Вагові коефіцієнти використані для відображення різного впливу на складність програми кожної функціональної групи. На думку автора метрики найбільшу вагу, що дорівнює трьом, має функціональна група С, так як вона впливає на потік управління програми. Вагові коефіцієнти інших груп розподіляються таким чином: $a_1 = 1$; $a_2 = 2$; $a_4 = 0.5$. Ваговий коефіцієнт групи Т не дорівнює нулю, оскільки "паразитні" змінні не збільшують складності потоку даних програми, але іноді ускладнюють її розуміння. З урахуванням вагових коефіцієнтів вираз набуде вигляду:

$$Q = P + 2M + 3C + 0.5T.$$

2.1.7 Метрики та їх вплив і аналіз ефективності використання

У таблиці надана зведена оцінка за базовими метриками в стилі:

ЩО - НАВІЩО - НА ЩО ВПЛИВАЄ

Таблиця 2 - Склад метрик, їх вплив і аналіз ефективності використання

Метрика	Навіщо потрібна	Впливає на...	Аналіз на основі статистичних даних (як тренд, так і прогноз))
Зусилля розробника при реалізації .	Наскільки ефективною є робота розробника.	Точність прогнозів оцінки трудомісткості при виконанні організацією типових запитів або запитів , які мало відрізняються	Можна аналізувати зусилля розробника у тимчасовому зрізі або у зрізі по релізів або проектам. Виявляти, на яких завданнях програміст повністю викладається, а які йому не до душі. Тренд дозволить менеджеру краще розуміти, хто і яких завданнях максимально ефективний при формуванні команди нового проекту, а також які підсистеми щодо складні, а які - прості.
Длина и объем программы		Оценку объема изменений	Увеличивается или уменьшается объем программы во времени. Используем для прогноза сложности на ранних этапах на основе

			статистики.
Анализ цикломатической сложности.		Оценку сложности изменений	Сложность растет или нет? Используем для прогноза сложности на ранних этапах на основе статистики.
Усилия программиста при разработке.	Для определения сложности реализации того или иного блока кода (класса, функции и т.д.)	Понимание того, насколько интеллектуально-затратной для разработчика была та или иная функция.	Анализируется увеличение или уменьшение усилий разработчика во времени. На предварительных этапах метрику можно использовать для прогноза.
Количество строк на реализацию требования.	Меряем общую температуру. Эта метрика принимается во внимание при анализе реализации и запроса.	Понимание КПД. Отслеживаем всплески.	Сигнал опасности при выявлении увеличения количества строк во время выполнения типового запроса. Используем для оценки сложности на ранних этапах на основе статистики.
Количество комментариев на единицу кода.	Код должен быть документирован. Если соотношен	Качество кода, его прозрачность.	Общая культура разработчиков растет или нет? Если растет – хорошо. Если нет – плохо. Если скачкообразно – соотносим

	не кода к комментар ию не 1:4, то разработч ик обязан доработать .		менеджеров\руководителей проектов со скачками. Выделяем сложные проекты, проблемные модули или подсистемы
Прочие количес твенные метрики (число функций, классов, файлов).	Отношени е новых функций к измененны м.	Количество добавленных, удаленных и измененных строк по отношению к предыдущей версии.	Глубокий анализ изменений по релизам (версиям, сборкам) дает понять: Количество изменений (на что угодно) – сколько раз один и тот же блок кода корректировался. Возможно выявить узкое место в программе: интенсивно меняющийся блок кода может влиять на общее качество программы (потенциальное место возникновения ошибок). Возможно, необходимо изменить архитектуру блока.
Плотност ь дефектов на единицу кода.	Количество о дефектов на 1-у строку кода	Производная метрика: количество строк/число дефектов.	Данная метрика более полезна для временной оценки: Плотность увеличивается от билда к билду, от версии к версии? Плотность дефектов по подсистемам (выявляем проблемную подсистему. В этом случае показатель почти наверняка будет
			коррелироваться с метрикой, отвечающей за интенсивность изменений участка кода, так как в этом месте наверняка «тонко»)

2.2. Попередня оцінка якості

2.2.1. Попередня оцінка якості на основі статистичних методів в залежності від етапу розробки програми

При використанні інтегрованих інструментальних засобів у компаній, що розробляють типові рішення (під цю категорію потрапляють так звані «інхаузери» - компанії, що займаються обслуговуванням основного бізнесу) з'являється можливість будувати прогнози складності програм, ґрунтуючись на зібраній статистиці. Статистичний метод добре підходить для вирішення подібних

типових завдань і практично не підходить для прогнозу унікальних проектів. У випадку унікальних проектів застосовуються інші підходи, обговорення яких знаходиться за рамками даного матеріалу.

Типові завдання як з рогу достатку падають на відділи розробки з бізнесу, бо попередня оцінка складності могла б сильно спростити завдання планування та управління, тим більше, що є накопичена база по проектах, в якій збережено не лише остаточні результати, а й всі початкові та проміжні.

Виділимо типові етапи в розробці програм:

- розробка специфікації вимог до програми;
- визначення архітектури;
- опрацювання модульної структури програми, розробка інтерфейсів між модулями.

Опрацювання алгоритмів;

- розробка коду і тестування.

Тепер спробуємо розглянути ряд метрик, що часто використовуються для попередньої оцінки на перших двох етапах.

2.2.2. Попередня оцінка складності програми на етапі розробки специфікацій вимог до програми

Для оцінки за результатами роботи даного етапу може бути використана метрика прогнозованого числа операторів $N_{\text{прогн}} програми$:

$$N_{\text{прогн}} = NF * N_{\text{од}}$$

Де: NF - кількість функцій чи вимог у специфікації вимог до розробляється програма;

$N_{\text{од}}$ - одиничне значення кількості операторів (середня кількість операторів, що припадають на одну середню функцію або вимога).

Значення $N_{\text{од}}$ - статистичне.

2.2.3. Попередня оцінка складності програми на етапі визначення архітектури

$$C_i = NI / (NF * N_{\text{од}} * KСЛ)$$

де NI - загальна кількість змінних, що передаються через інтерфейси між компонентами програми (також є статистичної);

$N_{\text{од}}$ -одиничне значення кількості змінних, що передаються через інтерфейси між компонентами (середнє число переданих через інтерфейси змінних, що припадають на одну середню функцію або вимога);

$KСЛ$ - коефіцієнт складності розробляється програми, враховує зростання одиничної складності програми (складності, що припадає на одну функцію або вимога специфікації вимог до програми) для великих і складних програм в порівнянні з середнім ПС.