# ТЕМА 16. МЕТРИКИ КОДУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У загальному випадку застосування метрик дозволяє керівникам проектів і підприємств вивчити складність розробленого або майбутнього проекту, оцінити обсяг робіт, стилістику програми що розробляється і зусилля, витрачені кожним розробником для реалізації того чи іншого рішення. Однак метрики можуть служити лише рекомендаційними характеристиками, ними не можна повністю керуватися.

## 1 Кількісні метрики

**Кількість рядків коду.** Найпоширенішою метрикою вихідного коду, що відображає розмір програмного проекту, є показник кількості рядків коду SLOC (Source Lines Of Code,). Не вдаючись в чергове перерахування недоліків даної метрики, розглянемо особливості її обчислення, а також існуючі різновиди та похідні.

Спочатку SLOC застосовувався в умовах, коли мови програмування були відносно прості за структурою і для них характерним було відповідність одного рядка коду одній команді мови. Згодом мови програмування еволюціонували, стали набагато гнучкіші, і ця відповідність для них більше не виконувалося – один фізичний рядок коду тепер міг містити кілька команд мови. З урахуванням цього виділяють і два основні різновиди показника SLOC:

– кількість фізичних SLOC (використовувані абревіатури: LOC, SLOC, KLOC, KSLOC, DSLOC) – визначається як загальна кількість рядків коду, з урахуванням коментарів і порожніх рядків (при вимірюванні показника на кількість порожніх рядків, як правило, вводиться обмеження – враховується їх кількість, що не перевищує 25% загальної кількості рядків у вимірюваному блоці коду). Літера "D" (Delivered) означає "поставлений" код, тобто тільки той, що увійшов до закінченого програмного продукту. Дуже часто виникає необхідність враховувати не тільки поставлений код, але і допоміжний або проміжний, який не входить до закінченого продукту, але що потрібен для реалізації проекту і вимагає певних витрат праці;

– кількість логічних SLOC (використовувані абревіатури: LSI, DSI, KDSI, де SI – Source Instructions) – визначається як кількість команд і залежить від використовуваної мови програмування. В тому випадку, якщо мова не допускає розміщення на одному

рядку кількох команд, кількість логічних SLOC буде відповідати числу "фізичних за винятком порожніх рядків та рядків коментарів. Якщо ж мова програмування підтримує розміщення кількох команд на одному рядку, то один «фізичний» рядок повинний бути врахований як декілька «логічних», якщо він містить більше однієї команди мови, що використовується.

Нерідко показник SLOC використовується на ранніх стадіях роботи над проектом з метою отримання попередніх оцінок його загального обсягу. І хоча завдання це досить складне, а про отримання точних оцінок не може бути й мови, для його вирішення є спеціальні підходи, які забезпечують прийнятний результат:

– оцінка показника SLOC за аналогією. Дана методика дозволяє отримувати оцінку показника для всього проекту, що розробляється, або окремих його складових за допомогою порівняння функціональних властивостей з існуючими аналогами. При оцінці показника за даною методикою підбирається близький за функціональністю замінник з відомим значенням SLOC, далі на його підставі визначається прогнозна оцінка з урахуванням відмінностей у функціональності, у мовах програмування, можливості оптимізації на основі коду аналога й інш.;

– оцінка показника SLOC «знизу-вгору» експертним методом. Цією методикою передбачено розподіл проекту на ієрархічну структуру завдань (Work Breakdown Structure, WBS), на підставі яких виконується експертне оцінювання показника, та підсумовується в результаті для всього проекту. Як правило, експертами надаються три оцінки (найбільш вірогідна, максимальна і мінімальна), а далі вони усереднюються за допомогою бета-розподілу (розраховується сума мінімальної, максимальної і чотириразової найбільш імовірної оцінки, яка потім ділиться на шість).

Для метрики SLOC є багато похідних метрик, що дозволяють оцінити окремі показники проекту. Основними серед яких є:

– кількість пустих рядків (Blank Lines Of Code, BLOC);

– кількість рядків, що містять коментарі (Comment Lines Of Code, CLOC);

– кількість рядків, що містять вихідний код і коментарі (Lines with Both Code and Comments, C&SLOC);

– кількість рядків, що містять декларативний вихідний код;

– кількість рядків, що містять імперативний вихідний код;

– відсоток коментарів (кількість рядків коментарів, помножене на 100 і поділена на кількість рядків коду);

– середня кількість рядків для функцій (методів);

– середня кількість рядків, що містять вихідний код для функцій (методів);

– середня кількість рядків для модулів;

– середня кількість рядків для класів.

Не зважаючи на всі недоліки цієї метрики, вона досі є ефективним інструментом вимірювання якості та є складовою багатьох інших метрик.

Стилістика програми. Іноді важливо не просто підрахувати кількість рядків коментарів, а визначити їх щільність, тобто визначити якість документованості коду. Для оцінки **стилістики програми** (F), використовують вираз

$$F = \sum_{i=1}^{n} F_i \,, \tag{1}$$

де $F_i = SIGN\left(N_{комм.ш} / N_i - 0{,}1\right)$ – стилістика і-го фрагмента програми, де Nкомм.і – кількість коментарів у і-м фрагменті, Ni – загальна кількість рядків коду в і-му фрагменті.

**Метрики Холстеда.** Моріс Холстед (Maurice H. Halstead) у 1977 запропонував групу метрик, побудованих на аналізі кількості рядків і синтаксичних елементів вихідного коду програми.

Оцінка складності програми (Halstead Difficulty, HDiff) оцінюється як:

$$Hdiff = (NUOprtr / 2) \times (NOprnd / NUOprnd) \,, \tag{2}$$

де *NUOprtr* – словник операторів (кількість унікальних операторів програми, також символи-роздільники, імена процедур і знаки операцій);

*NUOprnd* – словник операндів (кількість унікальних операндів програми);

*Noprtr* – загальна кількість операторів в програмі;

*Noprnd* – загальна кількість операндів в програмі.

На основі (2) пропонується оцінювати зусилля програміста при розробці за допомогою показника HEff (Halstead Effort):

$$HEff = HDiff \times HPVol\ , \tag{3}$$

де $HPVoc = NUOprtr + NUOprnd$ – словник програми (Halstead Program Vocabulary, HPVoc).

Крім того, для оцінювання програми можливо використовувати:

– довжина програми (Halstead Program Length, HPLen):

$$HPLen = Noprtr + Nopnrd\ ;$$

– обсяг програми (Halstead Program Volume, HPVol):

$$HPVol = HPLen \times log2\ HPVoc\ .$$

Незважаючи на теоретичну можливість отримання попередніх оцінок трудомісткості проекту з використанням показника *HEff*, метрики Холстеда, як правило, обчислюються вже на основі готового вихідного коду і застосовуються для отримання апостеріорних оцінок. Вони можуть бути використані для того, щоб запобігти надмірному ускладнення окремих структурних елементів програмного проекту і зростання пов'язаних із цим ризиків.

**Метрики Джилба.** Ці метрики показують складність програмного забезпечення на основі насиченості програми умовними операторами або операторами циклу. Дана метрика, не дивлячись на свою простоту, досить добре відображає складність написання і розуміння програми, а при додаванні такого показника, як максимальний рівень вкладеності умовних і циклічних операторів, ефективність цієї метрики значно зростає.

Абсолютна складність за Джилбом:

$$Cl = \sum N_{ifoprtr}\ , \tag{4}$$

де *Nifoprtr* – кількість умовних операторів.

Відносна складність за Джилбом:

$$Cl^* = \frac{CL}{N_{oprtr}}, \tag{5}$$

де *Noprtr* – загальна кількість операторів в програмі.

### 3. Метрики складності потоку керування програми

Наступний великий клас метрик, заснований вже не на кількісних показниках, а на аналізі керуючого графа програми, називається метрики складності потоку керування програм.

Перед тим як безпосередньо описувати метрики, для кращого розуміння буде наведемо визначення керуючого графу програми і спосіб його побудови.

Нехай представлена певна програма. Для даної програми будується орієнтований граф, що містить лише один вхід і один вихід, при цьому вершини графа співвідносять з тими ділянками коду програми, в яких є лише послідовні обчислення і відсутні оператори розгалуження і циклу, а дуги співвідносять з переходами від блоку до блоку і гілками виконання програми. Умова при побудові даного графа: кожна вершина досяжна з початкової, і кінцева вершина досяжна з будь-якої іншої вершини.

**Цикломатична складність програми.** Найпоширенішою оцінкою, заснованою на аналізі отриманого графа, є цикломатичне число Мак-Кейба, що визначається як:

$$V(G) = e - n + 2p, \tag{6}$$

де e – кількість дуг, n – кількість вершин, p – кількість компонентів зв‘язності.

Кількість компонентів зв‘язності графа можна розглядати як кількість дуг, що необхідно додати для перетворення графа в сильно зв'язний. Сильно зв'язним називається граф, будь-які дві вершини якого взаємно досяжні. Для графів коректних програм, тобто графів, що не мають недосяжних від точки входу ділянок і "висячих" точок входу і виходу, сильно зв‘язний граф, як правило, виходить шляхом замикання дугою вершини, що означає кінець програми, на вершину, що позначає точку входу в цю програму. По суті V(G) визначає кількість лінійно незалежних контурів в сильно доладному графі. Так

що в коректно написаних програмах p=1, і тому формула (6) для розрахунку цикломатичної складності набуває вигляду:

$$V(G) = e - n + 2 . \quad (7)$$

Як правило, при обчисленні цикломатичної складності логічні оператори не приймаються до уваги, допускається також спрощений підхід, згідно з яким власне побудова графа не проводиться, а показник визначається на підставі підрахунку кількісті операторів керуючої логіки (if, switch і т. д.) і можливої кількості шляхів виконання програми. Метрика цикломатичної складності може бути розрахована для модуля, методу та інших структурних одиниць програми.

В процесі аналізу значень показника для окремих структурних елементів можна виявити елементи з високим значенням показника (наприклад, нормальне значення показника для методу – не вище 5-7), що свідчить про складність їх керуючої логіки і, відповідно, високих трудовитратах на розробку, тестування і супровід.

Обчислення метрики в ході реалізації проекту (а при детальному проектуванні воно можливо ще на цьому етапі, не чекаючи стадії кодування) дозволяє вчасно визначити найбільш складні, що супроводжуються високими ризиками, структурні одиниці та вжити заходів щодо усунення ризиків за рахунок внесення корективів.

Для метрики цикломатичної складності існує безліч варіацій, зокрема:

– "модифікована" цикломатична складність – розглядає не кожне розгалуження оператора множинного вибору (switch, case), а весь оператор як єдине ціле;

– "сувора" цикломатична складність – містить логічні оператори;

– "спрощена" цикломатична складність – передбачає обчислення на основі не графа, а підрахунку керуючих операторів;

– "актуальна" складність – визначається як кількість незалежних шляхів, що проходить програма при тестуванні;

– метрика складності глобальних даних – обчислюється як цикломатична складність модуля та збільшується на кількість взаємозв'язків з глобальними даними.

В цілому, метрики цикломатичної складності є досить гарним показником, що дозволяє своєчасно припинити подальше ускладнення окремих складових проекту і

спростити їх, попередивши ймовірні проблеми з заплутаним і нестабільним кодом в майбутньому.

**Метрики похідні від цикломатичної складності.** Метрика цикломатичної складності стала основою для створення похідних і якісно нових метрик, таких як:

– інтервальна метрика Дж. Майерса (G. Mayers) – розраховується як різниця значень цикломатичної складності й кількості окремих умов плюс одиниця;

– метрика У. Хансена (W. Hansen) – визначається парою "цикломатична складність" і "кількість операторів";

– метрика Пивоварського – обчислюється на основі модифікованої цикломатичної складності і дозволяє враховувати структурованість програми, виявляючи код з поганою структурою, і багатьох інших, нерідко мають швидше за наукову, ніж практичну цінність:

$$N(G_i) = V^*(G_i) + \sum P_i, \tag{8}$$

де *V*(G)* – модифікована цикломатична складність, обчислена так як і *V(G)*, але з однією відмінністю: оператор CASE з n виходами розглядається як один логічний оператор, а не як n – 1 операторів;

*Pi* – глибина вкладеності і-ї предикатної вершини. Для підрахунку глибини вкладеності предикатних вершин використовується число "сфер впливу". Під глибиною вкладеності розуміється кількість всіх "сфер впливу" предикатів, які повністю утримуються у сфері даної вершини, або перетинаються з нею. Глибина вкладеності збільшується за рахунок вкладеності не самих предикатів, а "сфер впливу". Міра Пивоварського зростає при переході від послідовних програм до вкладених і далі до неструктурованих, що є її величезною перевагою перед багатьма іншими заходами даної групи.

**Топологічна міра Чена.** Ця метрика висловлює складність програми через кількість перетинів кордонів між областями, утвореними графом програми:

$$M(G_i) = (n(G_i), N, Q_0), \tag{9}$$

Цей підхід застосовується лише до структурованих програм, що допускає лише послідовне з'єднання керуючих конструкцій. Для неструктурованих програм міра Чена істотно залежить від умовних і безумовних переходів. У цьому випадку можна вказати верхню і нижню границі. Верхня – є m+1, де m – кількість логічних операторів при їх взаємної вкладеності. Нижня – дорівнює 2. Коли керуючий граф програми має тільки одну компоненту зв‘язності, захід Чена збігається з цикломатичною мірою Мак-Кейба.

**Метрики Харрісона і Мейджела.** Ці метрики враховують рівень вкладеності і довжини програми.

Кожній вершині присвоюється своя складність у відповідності з оператором, що вона зображує. Ця початкова складність вершини може обчислюватися будь-яким способом, включаючи використання заходів Холстеда. Виділимо для кожної предикатної вершини підграф, породжений вершинами, які є кінцями вихідних з неї дуг, а також вершинами, досяжними з кожної такої вершини (нижня межа підграфа), і вершинами, що лежать на шляхах з предикатної вершини в яку-небудь нижню межу. Цей підграф називається сферою впливу предикатної вершини.

Приведеною складністю предикатної вершини називається сума початкових або наведених складнощів вершин, що входять в її сферу впливу, плюс первинна складність самої предикатної вершини.

Функціональна міра (SCOPE) програми:

$$f_i = \sum \chi_i , \tag{10}$$

де χi – це складність вершини керуючого графа.

Функціональним ставленням (SCORT) називається відношення числа вершин в керуючому графі до його функціональної складності, причому з числа вершин виключаються термінальні:

$$f^*_i = \frac{N\chi_i}{f_i} , \tag{6.11}$$

де N – це кількість вершин керуючого графа.

SCORT може приймати різні значення для графів з однаковим цикломатичним числом.

**Метод граничних значень** так само заснований на аналізі керуючого графа програми. Для визначення даного методу необхідно ввести кілька додаткових понять.

Нехай $G = (V, E)$ – керуючий граф програми з єдиною початковою і єдиною кінцевою вершинами.

Кількість вершин, що входять до дуг, називається негативним ступенем вершини, а кількість вихідних з вершини дуг – позитивним ступенем вершини. Тоді набір вершин графа можна розбити на дві групи: вершини, у яких позитивна ступінь <=1; вершини, у яких позитивна ступінь >=2.

Вершини першої групи назвемо приймаючими вершинами, а вершини другої групи – вершинами відбору.

Кожна приймаюча вершина має наведену складність, рівну 1, крім кінцевої вершини, наведена складність якої дорівнює 0. Наведені складності всіх вершин графа G підсумовуються, утворюючи абсолютну граничну складність програми. Після цього визначається відносна гранична складність програми:

$$S_в = 1 - (v - 1) / S_a , \quad (12)$$

де Sв – відносна гранична складність програми, Sa – абсолютна гранична складність програми, v – загальна кількість вершин графа програми.

Існує **метрика Шнейдевинда,** що виражається через кількість можливих шляхів в керуючому графі.

**Міра Вудворда.** В основі цієї метрики кількість перетинів дуг керуючого графа. Так як в добре структурованій програмі таких ситуацій виникати не повинно, то дана метрика застосовується в основному в слабко структурованих мовах (Асемблер, Фортран). Точка перетину виникає при виході керування за межі двох вершин, які є послідовними операторами.

### 4. Метрики складності потоку керування даними

Наступний клас метрик – метрики складності потоку керування даних.

**Метрика Чепина.** Суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля з допомогою аналізу характеру використання змінних зі списку введення-виведення.

Вся безліч змінних, що складають перелік введення-виведення, розбивається на 4 функціональні групи :

– змінні, що використовуються для розрахунків і для забезпечення виведення (Р);

– модифіковані, або створювані всередині програми змінні (М);

– «керуючі» змінні – змінні, які беруть участь в керуванні роботою програмного модуля (С),

– "паразитні" змінні – змінні не використовуються в програмі, (Т).

Оскільки кожна змінна може виконувати одночасно кілька функцій, необхідно враховувати її в кожній відповідній функціональній групі.

Отже метрика Чепина має вигляд:

$$Q = a1 * P + a2 * M + a3 * C + a4 * T , \quad (13)$$

де a1, a2, a3, a4 – вагові коефіцієнти, що відображають різний вплив на складність програми кожної функціональної групи.

На думку автора метрики, найбільшу вагу, рівну 3, має функціональна група С, так як вона впливає на потік керування програми. Вагові коефіцієнти інших груп розподіляються наступним чином: a1=1, a2=2, a4=0.5. Ваговий коефіцієнт групи Т не дорівнює 0, оскільки "паразитні" змінні не збільшують складність потоку даних програми, але іноді ускладнюють її розуміння. З урахуванням вагових коефіцієнтів:

$$Q = P + 2M + 3C + 0.5T . \quad (14)$$

**Метрика Спена.** Ця метрика ґрунтується на локалізації звернень до даних усередині кожної програмної секції. Спен – це кількість тверджень, які містять цей ідентифікатор, між першою і останньою появою в тексті програми. Отже, ідентифікатор, що з'явився n раз, має спен, рівний n–1. При великому спені ускладнюється тестування та налагодження.

Метрика "модуль – глобальна змінна". Ця метрика, що зв'язує складність програм із зверненнями до глобальних змінних. Пара "модуль-глобальна змінна" позначається як (p, r), де p – модуль, що має доступ до глобальної змінної r. У залежності від наявності у програмі реального звернення до змінної r формуються два типи пар "модуль – глобальна змінна": фактичні та можливі. Можливе звернення до r за допомогою p показує, що область існування r містить у собі p.

Дана характеристика позначається Aup і говорить про те, скільки разів модулі up дійсно отримували доступ до глобальних змінних, а число Pup – скільки разів вони могли б отримати доступ.

Відношення кількості фактичних звернень до можливих визначається як:

$$Rup = Aup/Pup. \quad (15)$$

Ця формула показує наближену ймовірність посилання довільного модуля на довільну глобальну змінну. Очевидно, що чим вище ця ймовірність, тим вище ймовірність "несанкціонованої" зміни будь-якої змінної, що може істотно ускладнити роботу, пов'язану з модифікацією програми.

**Міра Кафура.** Вона створена на основі концепції інформаційних потоків. Для використання цієї міри вводяться поняття локального та глобального потоку.

Локальний потік інформації з модуля А до модуля В існує, якщо:

– модуль А викликає модуль В (прямий локальний потік);

– модуль В викликає модуль А і модуль А повертає значення, що використовується у модулі В (непрямий локальний потік)

– модуль С викликає модулі А і В та передає результат виконання модуля А до модуля В.

Глобальний потік інформації з модуля А через глобальну структуру даних D існує, якщо модуль А розміщує інформацію в D, а модуль В використовує інформацію з D.

На основі цих понять вводиться величина I – інформаційна складність процедури:

$$I = length*( fan\_in * fan\_out)^2 , \quad (16)$$

де length – складність тексту процедури (виміряються через метрику обсягу, типу метрик Холстеда (2-3), Маккейба (6), LOC і т.п.); fan_in – кількість локальних потоків входять всередину процедури плюс число структур даних, з яких процедура бере інформацію fan_out – кількість локальних потоків вихідних з процедури плюс число структур даних, які оновлюються процедурою.

Можна визначити інформаційну складність модуля як суму інформаційних складнощів процедур, що входять до нього.

Далі розглянемо інформаційну складність модуля щодо певної структури даних. Інформаційна міра складності модуля щодо структури даних:

$$J = NW * NR + NW * NRW + NRW * NR + NRW *(NRW-1), \quad (17)$$

де NW – кількість процедур, що тільки оновлюють структуру даних; NR – кількість процедур, що тільки читають інформацію з структури даних; NRW – кількість процедур, що і читають, і оновлюють інформацію в структурі даних.

**Міра Овієдо.** Суть її полягає в тому, що програма розбивається на лінійні непересічні ділянки – промені операторів, які утворюють керуючий граф програми.

Автор метрики виходить з таких припущень: програміст може знайти відношення між визначеними та використаними входженнями змінної всередині променя більш легко, ніж між променями; число різних визначальних записів у кожному промені більш важливо, ніж загальне число використовуваних входжень змінних в кожному промені.

Позначимо через R(i) безліч визначальних входжень змінних, які розташовані в радіусі дії променя i (визначальне входження змінної знаходиться в радіусі дії променя, якщо змінна або локальна в ньому і має визначальне входження, або для неї є визначальне входження в деякому попередньому промені, і немає локального визначення по шляху). Позначимо через V(i) множину змінних, які використовують входження i вже є в промені i. Тоді міра складності i-го променя визначається як:

$$DF(i) = \sum (DEF(v_j), \quad (18)$$

де $DEF(v_j)$ – визначає кількість входжень змінної vj з безлічі R(i),

$j = i...//V(i)//$, а //V(i)// – потужність безлічі V(i).

### 5. Метрики складності потоку керування і даних програми

Четвертим класом метрик є метрики, близькі як до класу кількісних метрик, класу метрик складності потоку керування програми, так і до класу метрик складності потоку керування даними (строго кажучи, цей клас метрик і клас метрик складності потоку керування програми є одним і тим же класом – топологічними метриками, але має сенс розділити їх у даному контексті для більшої ясності). Даний клас метрик встановлює складність структури програми як на основі кількісних підрахунків, так і на основі аналізу структур керування.

**Тестуюча М-Міра.** Тестуючою мірою M називається міра складності, що задовольняє наступним умовам:

1. Міра складності простого оператора дорівнює 1.

2. $M(\{F_1; F_2; -; F_N\}) = e_i^n M(F_p)$.

3. $M(if\ \ p\ then\ \ f1\ else\ \ f2) = 2\max(M(F1), M(F2))$.

4. $M(while\ \ p\ do\ F) = 2M(F)$.

Міра зростає з глибиною вкладеності й враховує протяжність програми. До тестуючої міри близько примикає міра на основі регулярних вкладень. Ідея цієї міри складності програм полягає в підрахунку сумарної кількості символів (операндів, операторів, дужок) у регулярному виразі з мінімально необхідною кількістю дужок, що описує керуючий граф програми. Всі міри цієї групи чутливі до вкладеності керуючих конструкцій і до протяжності програми. Проте зростає рівень трудомісткості обчислень.

**Зв'язаність модулів програми.** Якщо модулі сильно пов'язані, то програма стає важкомодифікованою і важкою в розумінні. Дана міра не виражається кількісно.

Види зв'язаності модулів:

- зв'язаність за даними – якщо модулі взаємодіють через передавання параметрів і при цьому кожен параметр є елементарним інформаційним об'єктом. Це найбільш бажаний тип зв'язаності (зчеплення);
- зв'язаність за структурою даних – якщо один модуль посилає іншому складовою інформаційний об'єкт (структуру) для обміну даними;
- зв'язаність за керуванням – якщо один модуль посилає іншому інформаційний об'єкт – флаг, призначений для керування його внутрішньою логікою;
- зв‘язаність за загальною областю, якщо модулі посилаються на одну і ту ж область глобальних даних. Зв'язаність (зчеплення) за загальною областю є небажаним, оскільки, по-перше, помилка в модулі, що використовує глобальну область, може несподівано проявитися в будь-якому іншому модулі; по-друге, такі програми важкі для розуміння, так як програмісту важко визначити, які саме дані використовуються конкретним модулем;
- зв'язаність за вмістом – якщо один з модулів посилається всередину іншого. Це неправильний тип зчеплення, оскільки повністю суперечить принципу модульності, тобто подання модуля у вигляді чорного ящика. Зовнішня зв'язаність – два модуля використовують зовнішні дані, наприклад комунікаційний протокол.

- зв'язаність за допомогою повідомлень – найбільш вільний вигляд зв'язаності, модулі безпосередньо не пов'язані один з одним, і взаємодіють через повідомлення, що не мають параметрів.

- підкласова зв'язаність – відношення між класом-батьком і класом-нащадком, причому нащадок пов'язаний з батьком, а батько з нащадком – ні.

- зв'язаність за часом – два дії згруповані в одному модулі лише тому, що через обставини вони відбуваються в один час.

Відсутність зв'язаності – модулі не взаємодіють між собою.

**Міра Колофелло.** Ще одна міра, що стосується стабільності модуля h(G) . Вона може бути визначена як кількість змін, які потрібно зробити в модулях, відмінних від модуля, стабільність якого перевіряється, при цьому ці зміни повинні стосуватися модуля, що перевіряється.

**Метрика Мак-Клура.** Виділяються три етапи обчислення цієї метрики:

1. Для кожної змінної обчислюється значення її функції складності C(i) за формулою:

$$C(i) = (D(i)* J (i))/ n , \quad (19)$$

де D(i) – величина, що стосується сфери дії змінної i, J(i) – міра складності взаємодії модулів через змінну i, n – кількість окремих модулів у схемі розподілу.

2. Для всіх модулів, що входять у сферу розбиття, визначається значення їх функцій складності M(P) за формулою:

$$M (P) = fp * X (P) + gp*Y (P) , \quad (20)$$

де fp і gp – відповідно, кількість модулів, безпосередньо передують і безпосередньо наступних за модулем P, X(P) – складність звернення до модулю P,Y(P) – складність керування викликом з модуля P інших модулів.

3. Загальна складність MP ієрархічної схеми розподілу програми на модулі задається формулою:

$$MP = \sum M (P) \quad (21)$$

Для всіх можливих значень P – модулів програми. Дана метрика орієнтована на добре структуровані програми, складені з ієрархічних модулів, які задають функціональну специфікацію і структуру керування. Також мається на увазі, що в кожному модулі одна точка входу і одна точка виходу, модуль виконує рівно одну

функцію, а виклик модулів здійснюється у відповідності з ієрархічною системою керування, яка задає ставлення виклику на безлічі модулів програми.

**Міра Берлингера.** Метрика, заснована на інформаційній концепції. Міра складності обчислюється як:

$$M = \sum f_i * \log_2 p_i , \quad (22)$$

де $f_i$ – частота появи і-го символу, $p_i$ – імовірність його появи.

Недоліком цієї метрики є те, що програма, що містить багато унікальних символів, але в малій кількості, буде мати таку ж складність як програма, що містить невелику кількість унікальних символів, але у великій кількості.

### 6. Об'єктно-орієнтовані метрики

У зв'язку з розвитком об'єктно-орієнтованих мов програмування з'явився новий клас метрик, також названий об'єктно-орієнтованими (ОО) метриками. У даній групі найбільш часто використовуваними є набори метрик Мартіна і набір метрик Чидамбера і Кемерера.

**Метрики Мартіна.** Для використання цих метрик необхідно ввести поняття категорії класів. У реальності клас може досить рідко бути використаний повторно ізольовано від інших класів. Практично кожен клас має групу класів, з якими він працює в кооперації, і від яких він не може бути легко відокремлений. Для повторного використання таких класів необхідно повторно використовувати всю групу класів. Така група класів сильно пов'язана і називається категорією класів. Для існування категорії класів існують наступні умови:

– класи в межах категорії класу закриті від будь-яких спроб змін всі разом. Це означає, що, якщо один клас повинен змінитися, всі класи в цій категорії з великою ймовірністю зміняться. Якщо будь- який з класів відкритий для деякого різновиду змін, вони всі відкриті для такого різновиду змін;

– класи у категорії повторно використовуються тільки разом. Вони настільки взаємозалежні і не можуть бути відокремлені один від одного. Таким чином, якщо робиться будь-яка спроба повторного використання одного класу в категорії, всі інші класи повинні повторно використовуватися з ним;

– класи у категорії поділяють деяку загальну функцію або досягають деякої спільної мети. Відповідальність, незалежність і стабільність категорії можуть бути виміряні шляхом підрахунку залежностей, які взаємодіють з цією категорією.

Для категорій класів можуть бути визначені три метрики :

1. Доцентрове зчеплення (Ca) – кількість класів поза певної категорії, що залежать від класів всередині цієї категорії.

2. Відцентрове зчеплення (Ce) – кількість класів всередині певної категорії, що залежать від класів поза цієї категорії.

3. Нестабільність:

$$St = Ce /(Ca + Ce) . \quad (23)$$

Ця метрика має діапазон значень [0,1]. Якщо St = 0, це вказує максимально стабільну категорію, а якщо St = 1, це вказує максимально нестабільну категорію.

Абстрактність категорії визначається наступним чином:

$$A = N_A / N_{All} ,$$

де $N_A$ – кількість абстрактних класів в категорії, $N_{All}$ – спільна кількість класів в категорії.

Значення цієї метрики змінюються в діапазоні [0,1], якщо A=0, то категорія цілком конкретна, а якщо A=1 – категорія повністю абстрактна.

Тепер на основі наведених метрик Мартіна можна побудувати графік, на якому відображено залежність між абстрактністю і нестабільністю. Якщо на ньому побудувати пряму, що задається формулою I+A=1, то на цій прямий будуть лежати категорії, що мають кращу збалансованість між абстрактністю і нестабільністю. Ця пряма називається головною послідовністю.

Далі можна ввести ще 2 метрики: Відстань до головної послідовності:

$$D = /( A + St -1) / sqrt(2)/$$

Нормалізованої відстань до головної послідовності:

$$Dn = / A + St - 2 /$$

Практично для будь-яких категорій вірно те, що чим ближче вони знаходяться до головної послідовності, тим краще.

**Загальні об’єктно-орієнтовані метрики.** До найбільш поширених ОО метрик відносяться:

– кількість виклику віддалених методів (Number Of Remote Methods, NORM). При її обчисленні проглядаються всі конструктори і методи класу і підраховується кількість викликів віддалених методів (не визначених у класі та його батьків);

– відгук на клас (Response For Class, RFC) – кількість методів, які можуть викликатися примірниками класу, обчислюється як сума локальних і віддалених методів;

– зважена насиченість класу 1 (Weighted Methods Per Class 1, WMPC1) – відображає відносну міру складності класу на основі цикломатичної складності кожного його методу. Клас з більш складними методами і великою їх кількістю вважається більш складним (при обчисленні метрики батьківські класи не враховуються);

– зважена насиченість класу 2 (WMPC2) – міра складності класу, заснована на тому, що клас з більшою кількістю методів і метод з більшою кількістю параметрів є більш складними (при обчисленні метрики батьківські класи не враховуються);

– недолік зв‘язності методів 1 (Lack Of Зчеплення Methods Of 1, LOCOM1) – відображає міру взаємозалежності методів (зв‘язність характеризує ступінь взаємодії між елементами окремого модуля і його насиченість). Чим вище значення даної метрики, тим більше взаємозалежними і, відповідно, складними вважаються методи;

– недолік зв‘язності методів 2 (LOCOM2) – відображає міру взаємозалежності класів. Обчислюється як процентне відношення методів, що не мають доступу до специфічних атрибутів класу, до всіх атрибутів даного класу. Клас, що може викликати значно більше методів, ніж рівні йому за рівнем, є більш складним;

– недолік зв‘язності методів 3 (LOCOM3) – вимірює ступінь розходження методів у класі за атрибутами. Низьке значення свідчить про хорошу декомпозицію в класі, що виражається в його простоті, зрозумілості та готовності до повторного використання. Високе значення – нестачу зв‘язності, що збільшує складність і підвищує ймовірність помилок в процесі розроблення.

**Метрики Чидамбера і Кемерера.** Ці метрики засновані на аналізі методів класу, дерева успадкування і т.д.

Метрика WMC (Weighted methods per class), сумарна складність всіх методів класу:

$$WMC = \sum_{i=1}^{n} c_i,$$

де $c_i$ – складність і-го методу, обчислена за будь-якою з метрик (Холстеда і т.д. в залежності від потрібного критерію), якщо у всіх методів складність однакова, то WMC дорівнює n.

Метрика *DIT (Depth of Inheritance tree)* – глибина дерева спадкування (найбільший шлях по ієрархії класів до цього класу від класу-предка), чим більше, тим краще, так як при більшій глибині збільшується абстракція даних, зменшується насиченість класу методами, однак при досить великій глибині сильно зростає складність розуміння і написання програми.

Метрика *NOC (Number of children)* – кількість нащадків (безпосередніх), чим більше, тим вище абстракція даних.

Метрика *CBO (Coupling between object classes)* – зчеплення між класами, показує кількість класів, з якими пов'язаний вихідний клас. Для даної метрики справедливі всі твердження, введені раніше для зв'язаності модулів, тобто при високому CBO зменшується абстракція даних і погіршується повторне використання класу.

Метрика *RFC (Response for a class)* – RFC=/RS/, де RS – відповідна множина класу, тобто безліч методів, які можуть бути потенційно викликані методом класу у відповідь на дані, отримані об'єктом класу. Тобто RS=(({M}({Ri}), i=1...n , де M – всі можливі методи класу, Ri – всі можливі методи, які можуть бути викликані і-м класом. Тоді RFC буде потужністю цієї множини. Чим більше RFC, тим складніше тестування та налагодження.

Метрика *LCOM (Lack of cohesion in Methods)* – недолік зчеплення методів. Для визначення цього параметра розглянемо клас C з n методами M1, M2, ... ,Mn, тоді {I1},{I2},...,In{} – безлічі змінних, що використовуються в цих методах. Тепер визначимо P – безліч пар методів, що не мають спільних змінних; Q – безліч пар методів, що мають загальні змінні.

Тоді

LCOM = / P /– / Q / .

Кількість пар методів розраховується як

$$C_m^2 = \frac{m!}{2(m-2)!},$$

де m – кількість методів класу.

Недолік зчеплення може бути сигналом того, що клас можна розбити на кілька інших класів або підкласів, так що для підвищення інкапсуляції даних і зменшення складності класів і методів краще підвищувати зчеплення.

**7. Гібридні метрики**

У завершенні необхідно згадати ще один клас метрик, що називаються гібридними. Метрики даного класу ґрунтуються на більш простих метриках і являють собою їх зважену суму.

**Метрика Кокола.** Вона визначається наступним чином:

$$H\_M = (M + R1 * M(M1) + ... + Rn * M(Mn)/(1 + R1 + ... + Rn)$$

де M – базова метрика, M*i* – інші цікаві підходи, R*i* – коректно підібрані коефіцієнти, M(Mi) функції.

Функції M(Mi) і коефіцієнти Ri обчислюються за допомогою регресійного аналізу або аналізу завдання для конкретної програми.

В результаті досліджень, автор метрики виділив три моделі для підходів: Маккейба, Холстеда і SLOC, де в якості базової використовується міра Холстеда. Ці моделі отримали назву "краща", "випадкова" і "лінійна".

**Метрика Зольновского, Сіммонса, Тейера.** Вона також являє собою зважену суму різних індикаторів. Існують два варіанти даної метрики:

– (структура, взаємодія, обсяг, дані) СУМА(a, b, c, d).

– (складність інтерфейсу, обчислювальна складність, складність введення/виводу, читабельність) СУМА(x, y, z, p).

Використовувані метрики в кожному варіанті вибираються залежно від конкретної задачі, коефіцієнти – в залежності від значення метрики для прийняття рішення в даному випадку.

Підводячи підсумок, хотілося б відзначити, що жодної універсальної метрики не існує. Будь-які контрольовані метричні характеристики програми повинні контролюватися або залежно один від одного, або залежно від конкретної задачі, крім

того, можна застосовувати гібридні заходи, однак вони так само залежать від більш простих метрик і також не можуть бути універсальними. Строго кажучи, будь-яка метрика – це лише показник, який сильно залежить від мови і стилю програмування, тому ні одну міру не можна зводити в абсолют і приймати будь-які рішення, ґрунтуючись тільки на ній.