Подібні властивості елементів можна задати або змінити в будь-який момент при проектуванні програми Створене додаток можна протестувати за допомогою емулятора і потім згенерувати відповідний файл. Сформований файл (з розширенням apk для Android і з розширенням xap для Windows Phone) автоматично зберігається на комп'ютері користувача.

У зв'язку з тим, що серверна частина програми використовується як репозиторій діаграм, клієнтська і серверна частини взаємодіють між собою за допомогою post-запитів (на відміну від get запитів, post запити передають всю необхідну інформацію в тілі запиту, а не в заголовку, і таким чином переважні для використання в системах, де необхідно передавати файли великої довжини).

Для можливості вільного доступу користувачів до розробленого конструктору, прототип дизайнера і емулятора викладений в «хмару» Amazon. В рамках даної дипломної роботи були реалізовані сайт, веб -сервер, app сервер, всі необхідні бази даних, а також забезпечена взаємозв'язок між різними компонентами системи (клієнтська частина, веб -сервер, генератори і сервер додатків). Розробка клієнтської [4] частини (дизайнера і емулятора) і генераторів не була частиною поставленого завдання і була проведена іншими людьми в рамках відповідних курсових і дипломних робіт. Тим не менше, їх опис наводиться тут для загального розуміння технології роботи реалізованого конструктора .

ЛІТЕРАТУРА

1. Burnstein, I. Practical software testing [Текст] / I. Burnstein. – Department of Computer Science. Illinois Institute of Technology Chicago, USA, 2002. – 732
2. Patton, R. Software Testing [Текст] / R. Patton. – Sams Publishing, USA, 2005 – 408
3. Fewster, M. Software Test. Automation Effective use of testexecution tools [Текст] / M. Fewster, D. Graham. – Biddies Ltd, Great Britain, 1994. – 591.
4. Myers, G. J. The Art of SoftwareTesting [Текст] / G. J. Myers. – John Wiley & Sons, Inc., Hoboken, New Jersey, Canada, 2004. – 255 с.

Д. Ю. Лучкін,
студент V курсу,
Харківський національний університет радіоелектроніки

Т. Б. Шатовська,
доцент,
Харківський національний університет радіоелектроніки

МЕТРИКИ ОЦІНКИ ЯКОСТІ ПРОГРАМНОГО КОДУ

Однією з тем у програмуванні є питання метрик коду програмного забезпечення. У великих програмних середовищах час від часу з'являються механізми підрахунку різних метрик. Хвилеподібний інтерес до теми так виглядає тому, що до цих пір в метриках не придумано головного – що з ними робити. Тобто навіть якщо якийсь інструмент дозволяє добре підрахувати деякі метрики, то що з цим робити далі найчастіше незрозуміло. Ця стаття – огляд найбільш відомих метрик коду програмного забезпечення.

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

Кількісні метрики. Перш за все, слід розглянути кількісні характеристики вихідного коду програм. Найелементарнішій метрикою є кількість рядків коду (SLOC). Дана метрика була спочатку розроблена для оцінки трудовитрат за проектом. Однак через те, що одна і та ж функціональність може бути розбита на декілька рядків або записана в один рядок, метрика стала практично непридатною з появою мов, в яких в один рядок може бути записано більше однієї команди. Тому розрізняють логічні і фізичні рядки коду. Логічні рядка коду – це кількість команд програми. Даний варіант опису так само має свої недоліки, так як сильно залежить від мови програмування і стилю програмування [1].

Крім SLOC до кількісних характеристик відносять також:

– кількість порожніх рядків;

– кількість коментарів;

– відсоток коментарів (відношення числа рядків, що містять коментарі до загальної кількості рядків, виражене у відсотках);

– середнє число рядків для функцій (класів, файлів);

– середнє число рядків, що містять вихідний код для функцій (класів, файлів);

– середнє число рядків для модулів.

Іноді додатково розрізняють оцінку стилістики програми (F). Вона полягає в розбитті програми на n рівних фрагментів і обчисленні оцінки для кожного фрагмента за формулою $Fi = SIGN (Nком.i / Ni – 0,1)$, де Nком.i – кількість коментарів у і-му фрагменті, Ni – загальна кількість рядків коду в і-му фрагменті. Тоді загальна оцінка для всієї програми буде визначатися таким чином: $F = \sum Fi$.

Ще одним типом метрик ПЗ, що відносяться до кількісних, є метрики Джілбі. Вони показують складність програмного забезпечення на основі насиченості програми умовними операторами або операторами циклу [2]. Дана метрика, не дивлячись на свою простоту, досить добре відбиває складність написання і розуміння програми, а при додаванні такого показника, як максимальний рівень вкладеності умовних і циклічних операторів, ефективність даної метрики значно зростає.

Метрики складності потоку управління даними. Метрика Чепіна: суть методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля за допомогою аналізу характеру використання змінних зі списку вводу-виводу. Всі

безліч змінних, складових список введення-виведення, розбивається на чотири функціональні групи.

Безліч «P» – що вводяться змінні для розрахунків та для забезпечення виведення. Прикладом може служити використовувана в програмах лексичного аналізатора змінна, що містить рядок вихідного тексту програми, тобто сама змінна не модифікується, а лише містить вихідну інформацію.

Безліч «M» – модифікуються або створювані всередині програми змінні.

Безліч «C» – змінні, що беруть участь в управлінні роботою програмного модуля (керуючі змінні).

Безліч «T» – не використовуються в програмі змінні. Оскільки кожна змінна може виконувати одночасно кілька функцій, необхідно враховувати її в кожній відповідній функціональній групі.

Далі вводиться значення метрики Чепіна:

Q = a1P + a2M + a3C + a4T, де a1, a2, a3, a4 – вагові коефіцієнти.

Вагові коефіцієнти використані для відбиття різного впливу на складність програми кожної функціональної групи. На думку автора метрики найбільшу вагу, що дорівнює трьом, має функціональна група C, так як вона впливає на потік управління програми. Вагові коефіцієнти решти груп розподіляються наступним чином: a1 = 1; a2 = 2; a4 = 0.5. Ваговий коефіцієнт групи T не дорівнює нулю, оскільки змінні не збільшують складності потоку даних програми, але іноді ускладнюють її розуміння. З урахуванням вагових коефіцієнтів вираз прийме вигляд:

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

Метрика Спен. Ця метрика ґрунтується на локалізації звернень до даних всередині кожної програмної секції. Спен – це число тверджень, які містять даний ідентифікатор, між його першим і останнім появою в тексті програми. Отже, ідентифікатор, що з’явився n раз, має Спен, рівний n – 1. При великому Спен ускладнюється тестування і налагодження.

Ще одна метрика, що враховує складність потоку даних – це метрика, 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. Залежно від наявності в програмі реального обігу до змінної r формуються два типи пар «модуль – глобальна змінна»: фактичні і можливі. Можливе звернення до r за допомогою p показує, що область існування r включає в себе 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 числа фактичних звернень до можливих визначається наступною формулою формулою: Rup = Aup / Pup.

Ця формула показує наближену ймовірність посилання довільного модуля на довільну глобальну змінну. Очевидно, що чим вище ця вірогідність, тим вище ймовірність «несанкціонованого» зміни якої-небудь змінної, що може істотно ускладнити роботи, пов’язані з модифікацією програми.

Метрики складності потоку керування програми. Наступний великий клас метрик, заснований на аналізі керуючого графа програми, називається метрики складності потоку керування програм.

Перед тим як безпосередньо описувати самі метрики, для кращого розуміння буде описаний керуючий граф програми і спосіб його побудови. Нехай представлена деяка 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 [3].

Найпоширенішою оцінкою, заснованою на аналізі отриманого графа, є цикломатична складність програми. Вона визначається як V(G) = e – n + 2p, де e – кількість дуг, n – кількість вершин, p – число компонент зв'язності. Число компонентів зв'язності графа можна розглядати як кількість дуг, які необхідно додати для перетворення графа в сильно зв'язний. Сильно зв'язковим називається граф, будь-які дві вершини якого взаємно досяжні. Для графів коректних програм, тобто графів, що не мають недосяжних від точки входу дільниць і «висячих» точок входу і виходу, сильно зв'язний граф, як правило, виходить шляхом замикання дугою вершини, що позначає кінець програми, на вершину, що позначає точку входу в цю програму. По суті V(G) визначає число лінійно незалежних контурів в сильно зв'язкового графі. Так що в коректно написаних програмах p = 1, і тому формула для розрахунку цикломатичної складності набуває вигляду: V (G) = e – n + 2.

Дана оцінка не здатна розрізняти циклічні та умовні конструкції. Ще одним істотним недоліком подібного підходу є те, що програми, представлені одними і тими ж графами, можуть мати абсолютно різні за складністю предикати.

Для виправлення даного недоліку Г. Майерсом була розроблена нова методика. Як оцінки він запропонував взяти інтервал [V (G), V (G) + h], де h для простих предикатів дорівнює нулю, а для n-х h = n-1. Даний метод дозволяє розрізняти різні по складності предикати, однак на практиці він майже не застосовується.

Ще одна модифікація методу Мак-Кейба – метод Хансена. Міра складності програми в даному випадку представляється у вигляді пари (цикломатична складність, число операторів). Перевагою даної міри є її чутливість до структурованості ПЗ.

Топологічна міра Чена висловлює складність програми через число перетинів кордонів між областями, утвореними графом програми. Цей підхід застосовується лише до структурованим програмами, 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 межі міри. Верхня – є m +1, де m – число логічних операторів при їх взаємної вкладеності. Нижня – дорівнює 2. Коли керуючий граф програми має тільки одну компоненту зв'язності, міра Чена збігається з цикломатичною мірою Мак– Кейба.

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. Даний клас метрик встановлює складність структури програми як на основі кількісних підрахунків, так і на основі аналізу керуючих структур.

Першою з таких метрик є тестуюча М-Міра. Міра зростає з глибиною вкладеності і враховує протяжність програми [4]. До тестуючої міри близько примикає міра на основі регулярних вкладень. Ідея цієї міри складності програм полягає в підрахунку сумарного числа символів (операндів, операторів, дужок) в регулярному виразі з мінімально необхідним числом дужок, що описує керуючий граф програми. Всі заходи цієї групи чутливі до вкладеності керуючих конструкцій і до протяжності програми. Однак зростає рівень трудомісткості обчислень.

Також мірою якості програмного забезпечення служить зв'язаність модулів програми. Якщо модулі сильно пов'язані, то програма стає важкою в розумінні та її трудно модефікуати. Дана міра не виражається чисельно. Види пов'язаності модулів наведені далі.

Зв'язаність за даними – якщо модулі взаємодіють через передачу параметрів і при цьому кожен параметр є елементарним інформаційним об'єктом. Це найбільш бажаний тип пов'язаності.

Зв'язаність за структурою даних – якщо один модуль посилає іншому складовою інформаційний об'єкт для обміну даними.

Зв'язаність з управління – якщо один посилає іншому інформаційний об'єкт – прапор, призначений для управління його внутрішньою логікою.

Модулі пов'язані з загальної області в тому випадку, якщо вони посилаються на одну і ту ж саму область глобальних даних. Зв'язаність з загальної області є небажаним, так як, 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 по вмісту – якщо один з модулів посилається всередину іншого. Це неприпустимий тип зчеплення, так як повністю суперечить принципу модульності, тобто подання модуля у вигляді чорного ящика.

Зовнішня зв'язаність – два модулі використовують зовнішні дані, наприклад комунікаційний протокол.

Зв'язаність за допомогою повідомлень – найбільш вільний вид пов'язаності, модулі безпосередньо не пов'язані один з одним, про повідомляються через повідомлення, які не мають параметрів.

Відсутність пов'язаності – модулі не взаємодіють між собою.

Зв'язаність за часом – дві дії згруповані в одному модулі лише тому, що зважаючи обставин вони відбуваються в один час.

Ще одна міра, що стосується стабільності модуля – міра Колофелло, вона може бути визначена як кількість змін, які потрібно провести в модулях, відмінних від модуля, стабільність якого перевіряється, при цьому ці зміни мають стосуватися перевіряється модуля.

Гібридні метрики. На завершення необхідно згадати ще один клас метрик, званих гібридними. Метрики даного класу ґрунтуються на більш простих метриках і являють собою їх зважену суму. Першим представником даного класу є метрика Кокопого. Вона визначається таким чином:

H_M = (M + R1 * M (M1) +... + Rn * M (Mn) / (1 + R1 +... + Rn)

Де M – базова метрика, Mi – інші цікаві заходи, Ri – коректно підібрані коефіцієнти, M (Mi) – функції.

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

В результаті досліджень, автор метрики виділив три моделі для заходів: Маккейба, Холстеда і SLOC, де в якості базової використовується міра Холстеда. Ці моделі отримали назву «найкраща», «випадкова» і «лінійна».

Підводячи підсумок, хотілося б відзначити, що жодної універсальної метрики не існує. Будь-які контрольовані метричні характеристики програми повинні контролюватися або в залежності один від одного, або в залежності від конкретного завдання, крім того, можна застосовувати гібридні заходи, проте 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.

ЛІТЕРАТУРА

1. http://www.ibm.com/developerworks/ru/library/r-rational_clear_case/
2. Липаев В.В. «Качество программных средств», М. – 2002
3. T.J. McCabe, «A complexity measure,» IEEE Transactions on Software Engineering, vol. SE-2, no. 4, pp. 308-320, December, 1976
4. Богданов Д.В., «Стандартизация жизненного цикла программных средств», СПб – 2000, 210 с.

Г. С. Павловська,
студентка VI курсу,
Національний технічний університет України
«Київський політехнічний інститут»

Ю. О. Олійник,
асистент кафедри автоматизованих систем
обробки інформації та управління,
Національний технічний університет України
«Київський політехнічний інститут»

ФОРМАЛЬНИЙ АПАРАТ ОПИСУ СПЕЦИФІКАЦІЇ ВИМОГ НА ОСНОВІ МОВИ UML

Процес опису специфікації вимог є дуже важливим етапом життєвого циклу програмного забезпечення (ISO/IEC 12207) та є обов’язковим елементом всіх відомих моделей життєвого циклу програмного забезпечення. Специфікація вимог є важливим елементом для розробки та тестування програмного забезпечення. Найбільш популярною мовою моделювання в області розробки програмного забезпечення на даний момент є мова UML, призначена для візуалізації, специфікування, проектування та документування програмних систем.