УДК 004.891.3: 004.3

О.В.ПОМОРОВА, Т.О.ГОВОРУЩЕНКО

Хмельницький національний університет, Україна

АНАЛІЗ МЕТОДІВ ТА ЗАСОБІВ ОЦІНКИ ЯКОСТІ ПРОГРАМНИХ СИСТЕМ

В статті висвітлюються: оцінка якості та складності програмного забезпечення (ПЗ) на етапі проектування, рівні складності ПЗ, метрики програм з точки зору їх точності на етапі проектування ПЗ, засоби автоматизації побудови метрик та модель зрілості можливостей ПЗ. Розглянуто проблему опрацювання результатів метричного аналізу ПЗ. Зроблено висновки про модель життєвого циклу ПЗ, прийнятну для оцінки якості на етапі проектування ПЗ, та про методи вимірювання показників якості на етапі проектування ПЗ. Визначено основні недоліки відомих методів і засобів оцінки якості програмних систем.

Ключові слова: *складність програмного забезпечення, якість програмного забезпечення, життєвий цикл програмного забезпечення, метрики програмного забезпечення, модель зрілості можливостей програмного забезпечення*

Вступ

Якість ПЗ - це характеристика ПЗ, яка відображає ступінь його відповідності вимогам. При цьому вимоги можуть трактуватись досить широко, що породжує цілий ряд незалежних визначень поняття якості. Згідно визначення ISO [1], якість - це ступінь відповідності приступніх характеристик вимогам. Згідно [2], якість - це повнота властивостей і характеристик продукту, процесу або послуги, які забезпечують здатність задовольняти оголошеним або передбачуваним потребам. Згідно [3], якість ПЗ - це ступінь, в якій воно володіє потрібною комбінацією властивостей.

Ключовим фактором забезпечення ефективного застосування програмних продуктів є ретельне оцінювання та досягнення високих значень показників якості. Постійне підвищення складності функцій, реалізованих програмами в інформаційних системах, безпосередньо призводить до збільшення їх обсягу та трудомісткості створення. Відповідно до змін складності програм зростає кількість виявлених та залишених в них дефектів та помилок, що відображається на їх якості. Програма обсягом в мільйони рядків тексту в принципі не може бути безпомилковою. Проблема виявлення та усунення помилок загострюється по мірі збільшення складності задач та програм, які їх вирішують, і загрожує катастрофами в галузях, що використовують ПЗ [4-6].

Проблема полягає в тому, щоб забезпечити потрібну якість функціювання ПЗ з врахуванням того, що деяка невідома кількість помилок та дефектів завжди залишається в складних комплексах програм, і повинна бути блокована або скорочена їх негативна дія до допустимого рівня. В зв'язку з цим стратегічна задача в життєвому циклі сучасного ПЗ - забезпечення якості програмних продуктів [7].

У відповідності до стандартів [8], забезпечення якості - це сукупність планованих та систематичних заходів, необхідних для впевненості в тому, що продукція або процеси задовольняють певним вимогам до якості. Система забезпечення якості застосовно до програмних засобів - це сукупність методів та засобів організації керуючих та виконавчих підрозділів підприємства, які беруть участь в проектуванні, розробці та супроводженні комплексів програм з метою надання їм властивостей, що забезпечують задоволення певних потреб замовників та споживачів при мінімальних або допустимих витратах ресурсів [7].

1. Оцінки якості та складності ПЗ на етапі проектування

Якість ПЗ визначається якістю методів та інструментальних засобів, які застосовувались для забезпечення всього їх життєвого циклу. На практиці важливо оцінювати якість програм не лише в завершеному вигляді, але й в процесі їх проектування і розробки.

Оціночна або прогнозована якість програмного продукту - характеристики, оцінені або передбачені для ПЗ на кожній стадії життєвого циклу (ЖЦ), які базуються на якості процесів та технології його забезпечення [7].

ЖЦ є моделлю створення і використання ПЗ, яка відображає його різні стани, починаючи з моменту виникнення необхідності в даному програмному продукті і закінчуючи моментом його повного виходу із вжитку у всіх користувачів.

Існує багато моделей життєвого циклу ПЗ, але в міжнародних стандартах як фундаментальні класифікуються три з них [7]: каскадна (водоспадна), інкрементна (поетапна), еволюційна (спіральна). Каскадна модель передбачає перехід на наступний етап після повного завершення робіт попереднього етапу. Основним недоліком каскадної моделі ЖЦ є можливість оцінювання якості і внесення зауважень замовником і користувачами лише після того, як робота над ПЗ буде повністю завершена. Поетапна модель з проміжним контролем - це ітераційна модель розробки ПЗ з циклами зворотнього зв'язку між етапами. В такій моделі міжетапний контроль та коригування забезпечують більшу гнучкість, надійність і якість ПЗ, хоча зібльшується період створення ПЗ. Спіральна модель ЖЦ перевіряє реалізованість технічних рішень на етапах аналізу вимог, визначення специфікацій та проектування шляхом створення прототипів додатків (програм, моделюючих роботу готової системи), які демонструються замовнику. Кожен виток спіралі відповідає створенню фрагмента або версії ПЗ, на ньому уточнюються мета і характеристики проекту, визначається його якість і плануються роботи наступного витка спіралі.

На етапі проектування важливо закласти цілу низку вимог по якості: вимоги до структури програмної системи (ПС); вимоги до навігації по ПС; вимоги до дизайну інтерфейсів користувача; вимоги до мультимедіа-компонентів ПС; вимоги по зручності (usability); технічні вимоги.

На етапі проектування формується відповідь на питання "Яким чином програмна система буде реалізовувати висунуті до неї вимоги?".

Інформаційні потоки етапу проектування ПЗ [9]: вимоги до ПЗ, представлені інформаційною, функційною та поведінковою моделями аналізу. Інформаційна модель описує інформацію, які повинна обробляти ПЗ на думку замовника. Функційна модель визначає перелік функцій обробки інформації та перелік модулів програмної системи. Поведінкова модель фіксує бажану динаміку системи (режими її роботи). На виході етапу проектування - розробка даних, розробка архітектури та процедурна розробка ПЗ.

Для одержання оцінки значень показників якості за стандартом [10] використовуються наступні методи: вимірювальний, реєстраційний, розрахунковий та експертний, а також комбінації цих методів. Вимірювальний метод базується на використанні вимірювальних та спеціальних програмних засобів для одержання інформації про характеристики ПЗ (обсяг ПЗ, кількість рядків коду, кількість операторів, кількість гілок в програмі, кількість точок входу/виходу та ін). Реєстраційний метод використовується при підрахунку часу, кількості збоїв або відмов, початку і кінця роботи ПЗ в процесі його виконання. Розрахунковий метод базується на статистичних даних, зібраних при проведенні випробувань, експлуатації та супроводженні ПЗ. Розрахунковими методами оцінюються показники надійності, точності, стійкості та ін. Експертний метод здійснюється групою експертів. Їх оцінка базується на досвіді та інтуїції, а не на безпосередніх результатах розрахунків або експериментів. Цей метод реалізується шляхом перегляду програм, кодів, супровідних документів, описів вимог до ПЗ групою експертів. Для цього встановлюються контрольовані ознаки, корельовані з одним або декількома показниками якості і включені в карти опитування експертів [11]. Метод застосовується при оцінці таких показників, як аналізованість, документованість, структурованість ПЗ та ін.

Отже, спіральна модель життєвого циклу дає можливість визначати якість вже на етапі проектування із застосуванням розрахункових та експертних методів вимірювання.

2. Рівні складності програмного забезпечення

За особливостями і властивостями життєвого циклу програм їх доцільно ділити на ряд класів і категорій, з яких найбільш розрізняються два великих класи - малі й великі [7].

Клас малих програм складають відносно невеликі програми, які створюються одним або декількома (3-5) програмістами. Малі програми створюються переважно для одержання конкретних результатів та аналізу відносно простих процесів. Вони не призначені для масового поширення, не мають конкретного незалежного замовника-споживача, не обмежуються замовником певною вартістю, трудомісткістю, вимогами заданої якості та термінами їх створення та не підлягають незалежному тестуванню, гарантуванню якості та сертифікації. До малих програм можна віднести програми, призначені для проміжних розрахунків, для потреб програміста і т.і.

Клас великих програм складають масштабні комплекси програм для складних систем управління та обробки інформації, які оформляються у вигляді програмних продуктів з гарантованою якістю. Великі програми мають великий розмір та високу вартість. Від замовника великої програми або програмної системи розробники повинні одержати конкретні вимоги до характеристик та функційності продукта. Від розробників проектів вимагаються гарантії високої якості, надійності функціювання та безпеки застосування компонентів і програмних систем. До великих програм відноситься системне програмне

забезпечення, а також деяке прикладне програмне забезпечення, наприклад, офісні пакети, графічні редактори і т.і. Великі програми називають ще програмними системами (ПС) або інформаційними системами (ІС).

Програмна складність характеризується довжиною програми або обсягом пам'яті ЕОМ, необхідної для розміщення ПЗ [12].

Структурна складність програм визначається кількістю взаємодіючих компонент, кількістю зв'язків між компонентами та складністю їх взаємодії [12].

Складність деякого міжмодульного зв'язку в процесі проектування можна характеризувати ймовірністю помилки при її формалізації та ступенем впливу цієї помилки на наступне функціювання модулів [12].

Складність програмних модулів характеризується конструктивною складністю створення оформленої компоненти програми і оцінюється з позиції складності внутрішньої структури та перетворення змінних в кожному модулі, а також інтегрально за деякими зовнішніми статичними характеристиками модулів [12].

Складність програми для систем реального часу переважно визначається допустимим часом відгуку, а для інформаційних систем - кількістю типів оброблюваних змінних [12].

3. Метрики програмного забезпечення

Згідно [13], метрика визначається як міра ступеня володіння властивістю, яка має числове значення. Взагалі, метрика ПЗ - це міра, яка дозволяє одержати числове значення деякої властивості ПЗ або його специфікацій. Отже, якщо мова йде про метрики складності програм, то це міри, які дають числове значення складності ПЗ.

Показники оцінки складності проекту є дуже важливими для одержання об'єктивних оцінок щодо проекту. Як правило, дані показники не можуть бути обчислені на ранніх стадіях роботи над проектом, оскільки вимагають, як мінімум, детального проектування. Однак ці показники важливі для одержання прогнозних оцінок тривалості та вартості проекту, оскільки безпосередньо визначають його трудомісткість.

Метрики складності програм прийнято розділяти на 3 основні групи [14]:

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

Метрики розміру програм базуються на визначенні кількісних характеристик, пов'язаних з розміром програми, і відрізняються відносною простотою. Метрики цієї групи орієнтовані на аналіз вихідного тексту програм, тому вони можуть використовуватись для оцінки складності проміжних продуктів розробки. До найбільш відомих метрик даної групи відносять кількість операторів програми, кількість рядків вихідного тексту, набір метрик Холстеда [14, 4].

Метрики складності потоку управління програм базуються на аналізі керуючого графу програми. Метрики другої групи теж можуть застосовуватись для оцінки складності проміжних продуктів розробки. Представником другої групи є метрика Маккейба [14].

Метрики складності потоків данпх програм базуються на оцінці використання, конфігурації та розташування даних в програмі. В першу чергу це стосується глобальних змінних. До даної групи відносять метрики Чепіна [14].

Розмірно-орієнтовані метрики (показники оцінки обсягу) прямо вимірюють програмний продукт і процес його розробки. Базуються такі метрики на LOC-оцінках (LOC - Lines of Code). Кількість рядків вихідного коду (Lines of Code – LOC, Source Lines of Code – SLOC) є найбільш простим і поширеним способом оцінки обсягу робіт по проекту. Ці метрики не універсальні, особливо це відноситься до показника LOC, який істотно залежить від використовуваної мови програмування.

В залежності від того, яким чином враховується вихідний код, виділяють 2 основних показники LOC [14, 4]: кількість «фізичних» рядків коду (LOC, SLOC, KLOC, KSLOC, DSLOC) — визначається як загальна кількість рядків вихідного коду, включаючи коментарі і порожні рядки; кількість «логічних» рядків коду (LSI, DSI, KDSI, де SI-source instructions) — визначається як кількість команд і залежить від використовуваної мови програмування. В тому випадку, якщо мова не допускає розташування декількох команд на одному рядку, то кількість "логічних" LOC буде відповідати кількості "фізичних", за винятком кількості порожніх рядків і рядків коментарів.

Метрика Холстеда належить до метрик, які обчислюються на основі аналізу кількості рядків та синтаксичних елементів вихідного коду програми.

Основу метрики Холстеда складають 4 вимірювані характеристики програми [14, 4]: NUOprtr (Number of Unique Operators) — кількість унікальних операторів програми, включаючи символи-роздільники, імена процедур і знаки операцій (словник операторів); NUOprnd (Number of Unique Operands) — кількість унікальних операндів програми (словник операндів); NOprtr (Number of Operators) — загальна кількість операторів в про-

грамі; NOpnd (Number of Operands) — загальна кількість операндів в програмі.

На основі цих характеристик розраховуються оцінки: словник програми, довжина програми, обсяг програми, оцінка її реалізації, складність її розуміння, трудомісткість кодування, інформаційний вміст, оптимальна модульність, складність програми ($HDiff = (NUOprtr / 2) \times (NOprnd / NUOprnd)$) [14].

Показник цикломатичної складності Маккейба є одним з найбільш розповсюджених показників оцінки складності програмних проектів. Цикломатичне число Маккейба показує необхідну кількість проходів для покриття всіх контурів сильнозв'язаного графа або кількість тестових прогонів програми, необхідних для вичерпного тестування за принципом «працює кожна гілка програми». Показник цикломатичної складності розраховується для модуля, метода та інших структурних одиниць програми.

Показник цикломатичної складності дозволяє не лише провести оцінку трудомісткості реалізації окремих елементів програмного проекту і скоригувати загальні показники оцінки тривалості та вартості проекту, але й оцінити зв'язані ризики і прийняти необхідні управлінські рішення.

Існує декілька модифікацій *метрик Чепіна*. Суть найпростішого з точки зору практичного використання і досить ефективного методу полягає в оцінці інформаційної міцності окремо взятого програмного модуля за допомогою аналізу характеру використання змінних зі списку введення-виведення. Вся множина змінних, яка складає список введення-виведення, розбивається на 4 функційні групи [14]: множина "P" - змінні, що вводяться для розрахунків та для забезпечення виведення; множина "M" - модифіковані або створювані всередині програми змінні; множина "C" - змінні, які приймають участь в управлінні роботою програмного модуля (керуючі змінні); множина "T" - не використовувані в програмі ("паразитні") змінні.

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

Значення метрики Чепіна: $Q = P + 2*M + 3*C + 0.5*T$.

Для оцінки складності програми використовуються й інші метрики [15]: метрики Джилба - кількість операторів циклу, кількість операторів умови, кількість модулів або підсистем, відношення кількості зв'язків між модулями до кількості модулів; метрика Шнадевида - кількість шляхів в графі керування; метрика Майєрса - інтервальна міра; метрика Хансена - пара (цикломатична кількість, кількість операторів); метрика Чена - топологічна міра; метрика Вудворда - кількість вузлів передач управління; метрика Кулика - кількість найпростіших циклів; метрика Хура -цикломатична кількість мереж Петрі; метрики Вітворфа, Зулевського - міра складності потоку керування, міра складності потоку даних; метрика Петерсона - кількість багатовходових циклів; метрики Харрісона, Мейджела - функційна кількість, функційне відношення, регулярні вирази; метрика Пивоварського - модифікована цикломатична міра складності; метрика Пратта - тестуюча міра; метрика Кантоне - характеристичні числа поліномів графу програми; метрика Мак-Клура - міра складності, заснована на кількості можливих шляхів виконання програми, кількості керуючих конструкцій та змінних; метрика Кафура - міра на основі концепції інформаційних потоків; метрика Схуттса, Моханті - ентропійні міри; метрика Коллофело - міра логічної стабільності програм; метрика Зольновського, Сімонса, Тейєра - зважена сума різних індикаторів; метрика Берлінгера - інформаційна міра; метрика Шумана - складність з позиції статистичної теорії мови; метрика Янгера - логічна складність з врахуванням історії обчислень; метрика Тая - покращення метрики Маккейба; метрика Кокола - комплексна метрика, заснована на більш простих; метрики зв'зності (зчеплення) - ступінь залежності кожного модуля від кожного з інших модулів за даними, за зразком, за управлінням, за зовнішніми посиланнями, за загальною областю, за змістом (кількісний показник - ступінь зчеплення).

Для оцінки складності програми на етапі розробки специфікації вимог до програми використовується метрика прогнозованої кількості операторів Nпрогн програми [14]: $N_{прогн} = NF * N_{од}$, де NF — кількість функцій або вимог в специфікації вимог до розроблюваної програми; $N_{од}$ - одиничне значення кількості операторів (середня кількість операторів, які доводяться на одну середню функцію або вимогу).

Оцінка складності програми на етапі визначення архітектури [14]: $C = \frac{NI}{NF * NI_{од} * K_{скл}}$, де NI - загальна кількість змінних, які передаються по інтерфейсах між компонентами програми (статистична); $NI_{од}$ – одиничне значення кількості змінних, які передаються по інтерфейсах між компонентами (середня кількість змінних, які передаються по інтерфейсах, що доводяться на одну середню функцію або вимогу); $K_{скл}$ — коефіцієнт складності розроблюваної програми, враховує ріст одиничної складності програми (складності, що доводиться на одну функцію або вимогу специфікації вимог до програми) для великих та складних програм в порівнянні з середнім показником складності.

В якості метрик процесу проектування використовуються:

- очікувана LOC-оцінка - за кожною функцією експерти надають краще, гірше та імовірне значення, тоді очікувана LOC-оцінка обчислюється за формулою:

$$LOC_{оч_i} = (LOC_{кращ_i} + LOC_{гірш_i} + 4 \times LOC_{імов_i}) / 6;$$

Є вільно поширювані інструменти очікуваної LOC-оцінки [16]; ця метрика на етапі проектування має прогнозоване значення;

- метрики складності ПЗ - метрики Холстеда та Маккейба; з метрик Холстеда на етапі проектування використовуються [9]: міра довжини модуля ($N = n_1 \log_2(n_1) + n_2 \log_2(n_2)$, де n_1 - кількість різних операторів, n_2 - кількість різних операндів) та обсяг модуля як кількість символів для запису всіх операторів і операндів тексту програми ($V = N \times \log_2(n_1 + n_2)$); для оцінки складності ПС, виходячи з топології внутрішніх зв'язків, Маккейб розробив метрику цикломатичної складності $V(G) = E - N + 2$, де E - кількість дуг, а N - кількість вершин в керуючому графі ПЗ; ці метрики на етапі проектування мають прогнозоване значення;

- метрики Чепіна - аналізують характер використання змінних зі списку введення, тобто оброблювану інформацію; ці метрики вже на етапі проектування мають точне значення;

- метрики зв'язності (зчеплення) - чим вище зв'язність модуля з іншими модулями, тим вища чутливість до внесення змін, тобто високий ступінь зв'язності (зчеплення) модуля з іншими модулями - це суттєвий недолік; ці метрики вже на етапі проектування мають точне значення;

- метрика Джилба - на етапі проектування можна тільки підрахувати кількість модулів та відношення кількості зв'язків між модулями до кількості модулів; ці складові метрики вже на етапі проектування мають точне значення;

- метрика Хансена - ця метрика на етапі проектування має прогнозоване значення;

- метрика Мак-Клура - метрика, спрямована на вимірювання архітектури системи; ця метрика вже на етапі проеткування має точне значення;

- метрика Кафура - метрика, заснована на врахуванні потоку даних; оскільки на етапі проектування вже є відомості про оброблювану інформацію, то й метрика має точне значення вже на етапі проектування;

- метрика Зольновського, Сіммонса, Тейєра - складова метрики (структура, взаємодія, обсяг, дані) обчислюється вже на етапі проектування і має точне значення;

- метрика звертання до глобальних змінних - пара (модуль, глобальна змінна); ця метрика вже на етапі проектування має точне значення;

- метрика Тая - складність програми визначається вимірюванням інформаційного потоку даних; вже на етапі проектування метрика має точне значення;

- метрика Кокола - комплексна метрика, яка, як правило, враховує значенн метрик Холстеда, Маккейба і LOC, отже, на етапі проектування має прогнозоване значення;

- метрика Вітворфа, Зулевського - складова метрики "міра складності потоку даних" має точне значення вже на етапі проектування;

- загальний час розробки і окремий час для кожної стадії - метрика має прогнозоване значення на етапі проектування;

- час модифікації моделей - метрика має точне значення на етапі проектування;

- час виконання робіт в процесі - метрика має прогнозоване значення на етапі проектування;

- кількість знайдених помилок при інспектуванні - метрика має точне значення на етапі проектування;

- прогнозована кількість операторів програми;

- прогнозована оцінка складності програми;

- очікувана вартість розробки кожної функції:

$$ВАРТІСТЬ_i = LOC_{оч_i} \times ВАРТІСТЬ_{рядка_i},$$

вартість рядка є константою і не змінюється від реалізації до реалізації;

- прогнозована продуктивність розробки кожної функції (на основі продуктивності аналогічних функцій в анлогічних програмних продуктах):

$$ПРОДУКТ_i = ПРОДУКТ_{ан_i} \times (LOC_{ан_i} / LOC_{оч_i})$$

- прогнозовані витрати на розробку кожної функції: $ВИТРАТИ_i = (LOC_{оч_i} / ПРОДУКТ_i)$;

- прогнозований функційний розмір FP - для обчислення функційного розміру: ідентифікуються очікувані від програмного додатку функції за критеріями International Function Point Users Group (IFPUG) [17]; для кожної виділеної функції порахувати кількість зовнішніх входів, кількість зовнішніх виходів, кількість зовнішніх запитів, кількість внутрішніх логічних файлів, кількість зовнішніх логічних файлів; кожен з факторів, визначених на попередньому кроці, множиться на коефіцієнт складності даного фактору в програмному додатку [17], такі добутки додаються за кожним з факторів; додати одержані для кожної функці суми (наближений функційний розмір); визначити ваги для 14 загальних характеристик проекту [11, 18] від 0 до 5 та знайти їх суму; обчислити уточнений функційний розмір за формулою:

$$\text{Уточн.функц.розмір} = \text{Наближений\_функц\_розмір} \times$$
$$\times[0.65 + 0.01 \times (\text{Сума\_загальних\_характеристик})]$$

Функційний розмір використовується як відносна метрика для порівняння з попередніми проектами, за його допомогою можна обчислити кількість рядків коду, що дозволяє визначити загальну трудомісткість та терміни проекту. Є вільно поширювані інструменти для обчислення функційного розміру [19];

- прогнозована оцінка трудовитрат та тривалості проекту - за моделлю Боема [18, 20] трудовитрати на розробку програмних додатків зростають швидше, ніж розмір додатків, для представлення даного співвідношення використовується експоненційна функція зі значенням показника, близьким до 1.12; тривалість проекту за моделлю Боема зростає експоненційно разом з докладеними до проекту зусиллями, однак в цьому випадку значення експоненти менше 1 і складає близько 0.35;
- прогнозована вартість перевірки якості;
- прогнозована вартість процесу розробки.

Різні метрики відображають різні аспекти складності ПС. Для всебічного врахування даних аспектів при оцінці ПС застосовується не одна метрика, а їх сукупність. Якщо було одержано декілька метрик, то кожне значення метрики множиться на відповідний ваговий коефіцієнт, встановлений експертним шляхом з огляду на домінуючі критерії якості відповідно до принципових задач, особливостей, функціонального призначення та властивостей ПС, а потім додаються всі показники для одержання комплексної оцінки рівня якості ПЗ або якості процесу проектування ПЗ. Найбільш актуально застосовувати досить великі сукупності метрик складності на етапі проектування, а в наступних етапах значення метрик фактично уточнюються.

Якщо вести мову про метрики процесу розроблення ПЗ, то однією з найпростіших метрик є ступінь виявлення дефектів для заданої фази виявлення і заданої фази появи [18]. Наприклад, "ступінь виявлення дефектів, рівний 0,2 дефекта на 100 вимог на стадії реалізації" означає, що на стадії реалізації 500 вимог, і в одній з них було виявлено дефект. Коли ступені виявлення дефектів стають рівними нормам організації, відбувається оцінка всього процесу в цілому, а не лише конкретного проекту. Результат такої метрики очевидний: більше дефектів виявляється на тому етапі, коли вони були допущені, а на більш пізніх етапах виявляється менше дефектів. Оскільки чим пізніше виявляється і виправляється дефект, тим дорожче це обходиться, даний факт може означати, що проект і процес розроблення виконаний краще.

Інші метрики процесу розроблення ПЗ [18]: кількість дефектів на тисячу рядків програмного коду, виявлених протягом 12 тижнів після завершення проекту; відхилення в розкладі на кожній фазі: $\frac{(\text{Фактична\_тривалість} - \text{Планова\_тривалість})}{\text{Планова\_тривалість}}$; відхилення у вартості: $\frac{(\text{Фактична\_вартість} - \text{Планова\_вартість})}{\text{Планова\_вартість}}$; відношення загального часу проектування до загального часу програмування - повинно бути не менше 50%; ступені появи і виявлення дефектів на деякій фазі; ймовірна норма дефектів, що залишились - відношення кількості дефектів на 100 рядків коду до норми організації кількості дефектів на 100 рядків коду; залишковий ступінь дефектності - кількість дефектних детальних вимог на 100 вимог піля завершення фази детальних вимог.

Процес розроблення ПЗ можна виміряти лише в порівнянні з даними по організації або по галузі, а самі по собі одержані числа ні про що не говорять [18].

Загальне покращення процесу розроблення ПЗ в першу чергу вимагає класифікації типів робіт і процесів, яка цілком залежить від компанії. Для кожного типу робіт визначається середня кількість дефектів на тисячу рядків вихідного коду до моменту завершення на етапах вимог, архітектури, детального проектування, реалізації для різних моделей життєвого циклу ПЗ - наприклад, каскадної, спіральної на 2-4 ітерації та спіральної на 5-10 ітерацій [18]. Для покращення процесу розроблення ПЗ використовуються кращі частини різних моделей.

В результаті аналізу метрик ПЗ можна визначити *основні проблеми метрології ПЗ*: відсутність загальноприйнятої номенклатури показників якості; неможливість проведення натурних випробувань програм на всій множині початкових даних; низька достовірність та недостатність інформації для одержання оцінок показників якості; недостатність засобів вимірювання метрик програми; відсутність обгрунтованих вимог до метричної інформації, виражених в числовому вигляді, які могли б підлягати перевірці; відсутність можливості інтерпретації одержуваних метрик і оцінок показників якості програм.

Отже, методи оцінки якості ПС, особливо на етапі проектування, на сьогодні є суб'єктивно залежними, оскільки використовуються експертні вагові коефіцієнти для метрик; порівняння значень метрик поточних проектів з попередніми (постає проблема, що робити, якщо проект принципово новий); немає загальноприйнятої номенклатури метрик; відсутні точні значення метрик, з якими можна було б порівняти поточні одержані значення.

4. Модель зрілості можливостей ПЗ. Автоматизація побудови метрик ПЗ

В останні роки запропонований та розвивається так званий "мовно-орієнтовний підхід (МОП) до вимірювань та оцінки ПЗ", який дозволяє створювати автоматичні, засновані на знаннях, програмні технології оцінки ПС, що базуються на однозначному, "прозорому" та семантично коректному описі понять прикладної області "якість програмного забезпечення" [23]. МОП до оцінки ПС полягає у використанні трьох рівнів узгоджених формальних мов, призначених для [23]:

1) точного визначення множини властивостей базового набору символів і конструкцій кожної використовуваної мови опису програмних продуктів, виходячи з визначень семантики і синтаксису цієї, точно визначеної мови. Ці властивості представляються в "вимірювальній моделі мови";

2) формального визначення метрик програмного продукту в термінах вимірювальних моделей мов опису цих продуктів;

3) формального визначення знань прикладної області "якість програмного забезпечення", специфікацій вимог до якості конкретних програмних продуктів в термінах формально визначених метрик і методів вимірювань програм, а також теорії вимірювань.

МОП підтримує всі відомі стандарти, моделі і метрики програмного забезпечення.

Зрозуміло, що порівнювати якість програмних продуктів без кількісних оцінок незручно й неможливо, тому доцільність застосування кількісних методів оцінки якості (метрик) очевидна. Питання, чи потрібні метрики ПЗ, давно відпало, наразі постає питання, як їх будувати. По мірі зростання актуальності програмних метрик на ринку з'являються різні "вимірювальні" програми. Одні з них досліджують характеристики проектів та ПЗ комплексно, інші орієнтовані на конкретні цілі - аналіз вихідного коду, розмірів і структури окремих модулів. На сьогоднішній день відомо більше тисячі видів метрик [24].

Для визначення загального рівня розвитку технологічних процесів в програмних організаціях розробили спеціальну систему оцінки зрілості технологічних процесів в організаціях, які випускають ПЗ, - модель Capability Maturity Model (CMM), засновану на так званих рівнях зрілості (maturity levels) [24]. Таких рівнів зрілості в CMM п'ять, і кожен з них характеризує певний ступінь якості програмних продуктів [24]:

1) початковий (initial) - відсутнє стабільне середовище розробки і супроводження; не витримуються терміни випуску продуктів; всі сили спрямовано на кодування і тестування програми (75% софтверних організацій);

2) повторюваний (repeatable) - жорстке керування, планування і контроль; акцент робиться на початкові вимоги, методи оцінки і конфігураційний менеджмент (15% софтверних організацій);

3) фіксований (defined) - процеси повністю документовані, стандартизовані і інтегровані в єдиний технологічний потік (8% софтверних організацій);

4) керований (managed) - намагання оцінити якість процесів і готового продукту кількісно; для контролю над процесами використовуються метрики (1.5% софтверних організацій);

5) оптимізований (optimizable) - намагання покращення роботи, керуючись кількісними критеріями якості; основна мета - випуск бездефектних продуктів, в яких помилки усунені ще на стадії внутрішнього тестування (0.5% софтверних організацій).

5-й рівень CMM вимагає постійного самостійного покращення процесу. Для покращення процесу керування проектом його ефективність вимірюється за допомогою метрик процесу [18]. Ці метрики дозволяють виміряти ефективність організації процесу, а також аналізу вимог, проектування, програмування і тестування.

Для одержання метричних даних і розрахунку показників застосовуються різні програми.

Для оцінки рівня компанії за шкалою CMM фірма Process Focus Software запропонувала програму CMM Live, яка представляє собою електронний довідник та експерт-радник по технологіях CMM; фірма Software Productivity Center - програмний пакет SoftGuide, орієнтований на розробників ПЗ, які намагаються покращити якість своїх процесів. Німецький національний дослідницький центр по інформаційних технологіях (GMD) розробив проект SCOPE*PROCEPT [24], який охоплює інформаційні моделі процесів створення ПЗ, включаючи методи оцінки якості на основі метрик. До складу SCOPE*PROCEPT входить декілька компонентів: специфікація процесів програмування і тестування (ProcePT), інжінірінг моделей якості (Model Y7), вимірювання характеристик (метрик) ПЗ (SMV) - кількісні дослідження характеристик програм; інтегроване середовище для вимірювання характеристик компіляторів (CISM); моделювання процесів виробництва (SPM) - оцінка якості процесів розробки на основі кількісних показників.

Програми набувають високої якості не стільки в результаті комплексного тестування кінцевого продукту, скільки в процесі його розроблення. Якщо в методології створення ПЗ закладено "відловлювання" помилок на всіх стадіях виконання проекту, то проект буде практично безпомилковим. Корпорація IBM пропонує методологію створення складних

програмних систем, яка має назву Cleanroom Software Engineering [25]. Вона дозволяє колективам розробників планувати, вимірювати, специфікувати, проектувати, кодувати, тестувати та сертифікувати програмні продукти. Інструментом автоматизованого тестування та оцінки надійності ПЗ в методології Cleanroom є середовище Cleanroom Certification Assistant, яке використовує статистичні результати тестування для підрахунку метрик надійності ПЗ математичними методами.

На ринку існує багато продуктів, які дозволяють автоматизувати процес верифікації: Logiscope, TestCenter, Purify, IBM Rational Software Group, Hindsight, EzCover.

Пакет Logiscope - це набір програм (TestChecker, RuleChecker, Audit), які проводять всебічне тестування створюваного ПЗ і покращують його якість. В основі продукту лежить ідея аналізу вихідного коду. Остання його версія здатна обробляти тексти програм, написані більш ніж на 80 мовах програмування. Пакет Logiscope [25] призначений для якісної оцінки кодів та пошуку місць, де поява помилок найбільш ймовірна. Після аналізу коду Logiscope формує масу різноманітної метричної інформації у вигляді кількісних показників (більше 200 типів метрик) про код, його позитивні та негативні сторони, генерує повний звіт, що дозволяє робити висновки про якість коду.

Спеціально для професійних програмістів мовами C і C++ призначена *програма TestCenter* компанії CenterLine. Із статистичних даних відомо, що при звичайному тестуванні перевіряється лише 40-50% загального коду програм. Пакет TestCenter організовує глобальне тестування ПЗ та інтегрує тестування в процес розробки.

Компанія Pure Software, провідний виробник автоматизованих інструментальних засобів створення якісного ПЗ, пропонує розробникам *систему Purify*, яка дозволяє виявляти різноманітні помилки програм, включаючи помилки виконання (runtime errors) та витоку пам'яті, дозволяє детально контролювати доступ до пам'яті і виявляти такі помилки, як використання неініціалізованих змінних, некоректні операції malloc/free, виходи за межі масивів, невірна робота з вказівникапми, стекові помилки.

Пакет *IBM Rational Software Group* [26] пропонує наступні продукти тестування: IBM Rational Purify (відстежування помилок часу виконання, які важко виявляються - витік пам'яті, виходи за межі масивів); IBM Rational PureCoverage (відстежування коду тестованого додатку, автоматизація процесу вимірювання метрик повноти тестування), IBM Rational Quantify (аналіз продуктивності працюючого додатку), IBM Rational Robot (автоматизація запису та відтворення сценаріїв тестів, тестування функційності та продуктивності системи), IBM Rational TestManager (планування тестів), IBM Rational TestFactory (автоматизація процесу тестування графічних компонентів), IBM Rational ManualTest (планування тестів, які не підлягають автоматизації), IBM Rational ClearQuest (управління запитами на зімни, утворення сховища дефектів, знайдених при тестуванні), IBM Rational RequisitePro (керування вимогами).

Програма Hindsight компанії IntegriSoft [27] аналізує вихідний код, проводить вимірювання вихідного коду і обчислює значення метрик програмного продукту для їх використання при оцінці якості. Обчислюються такі метрики, як цикломатична складність, складність даних, метрики Холстеда, складність архітектури.

Інструмент EzCover [27] - це інструмент тестового покриття. Він проводить вимірювання програмного продукту та обчислює наступні метрики: цикломатична складність, видозмінена складність, складність даних, розгалуження за входом, розгалуження за виходом, кількість рядків порожніх, з коментарями, виконуваних.

Програма IBM Rational ClearCase [28] дозволяє точно визначати поточний стан проекту, видавати завдання, контролювати їх виконання і відстежувати стан проекту за такими метриками, як кількість запитів в роботі, кількість версій в розробці і кількість дефектів.

До *загальних недоліків згаданих засобів оцінки якості* можна віднести: суб'єктивну залежність вибору метрик, які буде формувати засіб автоматизації побудови метричної інформації; суб'єктивність інтерпретації метрик, адже точні (еталонні) значення метрик, з якими можна було б порівняти одержані метрики, відсутні; спрямованість засобів автоматизації побудови метричної інформації на тестування та метрологію готового вихідного коду, а не на прогнозування і розрахунки метрик програмного забезпечення на етапі проектування, коли готового продукту ще немає, а є лише інформаційна, функційна та поведінкова моделі аналізу вимог до ПЗ.

5. Опрацювання результатів метричного аналізу

На основі аналізу метрик можна зробити висновки про рівень якості вихідного коду [9, 22]: обчислити ймовірності виникнення помилок в окремих модулях, ступені зв'язності модулів та інші параметри. Однією з проблем аналізу метрик вихідного коду є складність інтерпретації обчислених величин. Середні значення метрик можуть значно відрізнятись для проектів різного роду. Наприклад, значення метрики "цикломатична складність", нормаль-

не для бібліотеки обчислювальних алгоритмів, невиправдано високе для клієнтського додатку.

Статистичний аналіз метричної інформації передбачає:

1) аналіз метрик за релізами: накопичення статистичної інформації метрик складності і якості ПЗ служить основою для управління складністю і якістю ПЗ в наступних проектах. Так, метрики довжини і обсягу програми дають інформацію про збільшення чи зменшення обсягу програми в часі. Метрика цикломатичної складності показує, чи зростає складність від релізу до релізу. Метрика кількості рядків на реалізацію вимоги попереджає про виявлення збільшення кількості рядків під час виконання типового запиту чи під час реалізації типової вимоги. До того ж ця метрика дозволяє програмісту або менеджеру проекту спрогнозувати кількість рядків коду при використанні типових вимог і функцій. Вони використовуються для прогнозу складності на ранніх етапах на основі статистики. Глибокий аналіз змін по релізах інших кількісних метрик (кількість функцій, класів, файлів) дає можливість виявити вузьке місце в програмі - блок коду, що інтенисвно змінюється - як потенційне місце виникнення помилок.

2) аналіз метрик за замовниками: якщо виділяються окремі гілки або функції програми для конкретного замовника. Потрібно зберігіати інформацію про всі зміни коду для всіх замовників.

3) вибірка проектних даних за певними критеріями - проектами, програмами, замовниками і т.п.

Для визначення задовільності чи незадовільності досягнутого рівня складності ПС використовується відношення розрахункового значення метрики до базового (статистичного) значення метрики, взятого зі статистичних даних попередніх проектів. Якщо таке відношення близьке до 1 або менше 1, то можна зробити висновок про задовільний рівень складності ПС за аналізованою метрикою. При значенні відношення, яке значно перевищує 1, можливо, слід виконати попередні роботи заново. При цьому слід враховувати змістовний вміст незадовільної метрики, щоб коригувати відповідні аспекти програмного продукту.

Для визначення впливу конкретних значень окремих метрик на загальну складність ПС виконується інтегральна оцінка складності [21]. Можливі 2 варіанти інтегральної оцінки: 1) інтегральну відносну складність ПС можна визначити як середнє арифметичне відносних результатів (відношень розрахункового значення метрики до базового) всіх метрик, якщо відсутня апріорна інформація про вплив результатів конкретної метрики на загальну складність проміжного або кінцевого продукту; 2) інтегральна складність ПС визначається як середньозважена сума одержаних значень метрик (сума ваг всіх метрик дорівнює 1), якщо наявна статистична інформація про сутпінь впливу значень конкретних метрик на інтегральну складність ПС.

При статистичному аналізі метричної інформації основною проблемою є неможливість опрацювання метрик для принципово нового проекту.

Важливість якості ПЗ збільшує інтерес до нових методів, які використовуються в побудові моделей якості ПЗ для прогнозування атрибутів якості. Одним з таких методів є метод, що базується на штучній нейронній мережі (ШНМ). В [29] показано використання ШНМ для прогнозу якості ПЗ на основі об'єктно-орієнтованих метрик. Залежною змінною в такому досліді були роботи по технічному обслуговуванню. Незалежними змінними були головні компоненти 8 об'єктно-орієнтованих метрик. Результат показав, що середня абсолютна величина відносної похибки був 0.265 для моделі з ШНМ. Тому можна зробити висновок, що ШНМ можуть бути корисні в побудові моделі якості ПЗ. В [30] доведено застосовність байєсових мереж в якості інструменту підтримки для числової оцінки ПЗ в реальному масштабі часу, особливо для додатків з особливими вимогами щодо безпеки. Байєсові мережі можуть також відігравати важливу роль при прийнятті рішення про сертифікацію ПЗ.

Очевидно, що проблема оцінювання результатів метричного аналізу програмних продуктів з використанням інтелектуальних методів (зокрема з використанням ШНМ) досліджувалась мало, причому лише для конкретних видів метрик (об'єктно-орієнтовані метрики) і для конкретних типів програмного забезпечення (об'єктно-орієнтоване ПЗ та ПЗ з особливими вимогами щодо безпеки). Отже, актуальними є подальші дослідження на предмет можливості використання ШНМ для прогнозу якості ПЗ на основі аналізу інших типів метричної інформації для різних типів ПЗ.

Висновки

Для визначення якості ПЗ на етапі проектування, найбільш прийнятною є спіральна модель життєвого циклу ПЗ. З опису методів вимі-рювання показників (метрик) якості зрозуміло, що на етапі проектування ПЗ можуть бути задіяні лише розрахункові та експертні методи вимірювання, ос-кільки неможливо виміряти жодної характеристики ще не розробленого ПЗ і неможливо реєструвати моменти процесу виконання ще не існуючого ПЗ.

Незважаючи на численні дослідження програмних метрик, в цій галузі залишається *багато неви-*

рішених питань. По-перше, лише 1.5% софтверних організацій намагаються оцінити якість процесів і готового продукту кількісно, за допомогою метрик і лише 0.5% софтверних організацій намагаються покращити роботу, керуючись кількісними критеріями якості з метою випуску бездефектних продуктів [22]. По-друге, технологія вимірювання якості ще не досягла зрілості, оскільки лише 0.5% софтверних організацій знаходяться на оптимізованому, зрілому рівні моделі CMM. По-третє, відсутні єдині стандарти на метрики, створено більше тисячі метрик, тому кожен постачальник "вимірювальної" системи пропонує власні способи оцінки якості і відповідно метрики. По-четверте, існує проблема складності інтерпретації величин метрик. Саме через невирішеність цих питань поки що неможливо створити бездефектне високоякісне ПЗ.

Приймаючи до уваги результати аналізу методів та засобів оцінки складності ПС, можна зробити висновок, що перспективним напрямком досліджень є розроблення інтелектуальних систем, які: 1) обчислюватимуть розрахунковими та експертними методами точні або прогнозовані значення метрик програмного забезпечення вже на етапі проектування; 2) не лише будуватимуть метрики, але й на основі одержаних значень метрик надаватимуть рекомендації і висновки про розроблюване програмне забезпечення, аналізуватимуть і опрацьовуватимуть результати метричних оцінок, оскільки для більшості користувачів як метрики, так і їх конкретні значення ні про що не говорять.

Література

1. *ISO 9001:1994 Quality systems - Model for quality assurance in design, development, production, installation and servicing*
2. *ISO 8402:1994 Quality management and quality assurance*
3. *1061-1998 IEEE Standard for Software Quality Metrics Methodology*
4. *Скляр В.В. Оценка качества и экспертиза программного обеспечения. Лекционный материал. - Харьков: НАУ "ХАИ", 2008. - 204 с.*
5. *Майерс Г. Надежность программного обеспечения: Пер. с англ. – М.: Мир, 1980. – 360 с.*
6. *Myers G.J. The Art of Software Testing. – New York: John Wiley and Sons, 1979. – 312 pp.*
7. *Липаев В.В. Программная инженерия. Методологические основы. - М.: ТЕИС, 2006. - 608 с*
8. *Сборник действующих международных стандартов ИСО серии 9000. Т.1, 2, 3. - М.: ВНИИКИ, 1998.*
9. *Орлов С.А. Технологии разработки программного обеспечения. Разработка сложных программных систем: Учебник для ВУЗов - СПб.: Питер, 2004. - 527 с.*
10. *ДСТУ 3230–1995. Управление качеством и обеспечение качества. Термины и определения*
11. *Петрухин В.А., Лаврищева Е.М. Методы и средства инженерии программного обеспечения // http://www.intuit.ru/department/se/swebok/10/1.html*
12. *Ковалевская Е.В. Материалы к курсу "Метрология, качество и сертификация ПО" - М.: Московский государственный университет экономики, статистики и информатики, 2002. - 38 с.*
13. *IEEE Standard Glossary of Software Engineering Terminology /IEEE Std 610.12-1990*
14. *Новичков А., Шамрай А., Черников А. Метрики кода и их практическая реализация в Subversion и ClearCase. Часть 1 - метрики // http://cmcons.com/articles/CC_CQ/dev_metrics/mertics_part_1/*
15. *http://kapustin_andrey.boom.ru/Materials/Metrics2.htm*
16. *http://www.construx.com*
17. *International Function Point User's Group, http://www.ifpug.org/*
18. *Брауде Э. Технология разработки программного обеспечения - СПб.: Питер, 2004. - 655 с.*
19. *International Function Point User's Group reference to function point spread-sheets, http://ifpug.org/home/docs/freebies.html*
20. *Boehm B. Software Engineering Economics - NJ: Prentice Hall, 1981. - 392 p.*
21. *Бахтизин В.В., Глухова В.А. Применение метрик сложности при разработке программных средств // http://www.giac.unibel.by/docs/pdf/1-2005/s10-1-2005.pdf*
22. *Липаев В.В. Выбор и оценивание характеристик качества программных средств: Методы и стандарты - М.: Синтег, 2001 - 224 с.*
23. *Коган Б.И. Автоматизация оценивания качества программного обеспечения // http://www.febras.ru/~conf/seminar/kogan.html*
24. *http://www.rol.ru/news/it/press/cwm/25_96/teh.htm*
25. *http://www.interface.ru/home.asp?artId=18833*
26. *http://software-testing.ru/library/vendors/156-ibm-rational*
27. *Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. - М.: – Издательство "Лори", 2003. – 568 с.*
28. *http://cmcons.com/tech_rational/IBM_Rational_ClearCase*
29. *K. K. Aggarwal, Yogesh Singh, Arvinder Kaur, and Ruchika Malhotra. Application of Artificial Neural Network for Predicting Maintainability using Object-Oriented Metrics // PROCEEDINGS OF WORLD ACADEMY OF SCIENCE, ENGINEERING AND TECHNOLOGY VOLUME 15 OCTOBER 2006 ISSN 1307-6884 // http://www.waset.org/pwaset/v15/v15-52.pdf*

30. Janusz Zalewski , Andrew J. Kornecki, Henry L. Pfister. Numerical Assessment of Software Development Tools in RealTime Safety Critical Systems Using Bayesian Belief Networks // http://www.proceedings2006.imcsit.org/pliks/194.pdf

Надійшла в редакцію 00.00.2009

Рецензент: доктор технічних наук, професор, завдіувач кафедри системного програмування Локазюк В.М., Хмельницький національний університет, м.Хмельницький

АНАЛИЗ МЕТОДОВ И СРЕДСТВ ОЦЕНКИ КАЧЕСТВА ПРОГРАММНЫХ СИСТЕМ

О.В. ПОМОРОВА, Т.О. ГОВОРУЩЕНКО

В статье освещаются: оценка качества и сложности программного обеспечения (ПО) на этапе проектирования, уровни сложности ПО, метрики программ с точки зрения их точности на этапе проектирования ПО, средства автоматизации построения метрик и модель зрелости возможностей ПО. Рассмотрена проблема обработки результатов метрического анализа. Сделаны выводы о модели жизненного цикла, приемлемой для оценки качества ПО на этапе проектирования ПО, и о методах измерения показателей качества на этапе проектирования ПО. Определены основные недостатки известных методов и средств оценки качества программных систем.

Ключевые слова: *сложность программного обеспечения, качество программного обеспечения, жизненный цикл программного обеспечения, метрики программного обеспечения, модель зрелости возможностей программного обеспечения.*

ANALYSIS OF SOFTWARE SYSTEM QUALITY VALUATION TECHNIQUES AND MEANS

O.V. POMOROVA, T.O. GOVORUSHCHENKO

In article software quality and complexity valuation on the design stage, software complexity levels, software metrics in the view of their exactness on the design stage, computer-aided metrics construction means and software capability maturity model are described. Problem of metric analysis results processing was viewed. The conclusions about life cycle model, acceptable for software quality evaluation on the design stage, and about quality coefficients measuring procedures on the design stage were made. Defects of known software system quality valuation techniques and means were specified.

Key words: *software complexity, software quality, software life cycle, software metrics, software capability maturity model.*

Поморова Оксана Вікторівна - доктор технічних наук, доцент, професор кафедри системного програмування, Хмельницький національний університет, м.Хмельницький, Україна, e-mail: **o.pomorova@gmail.com**, **kism@beta.tup.km.ua**, тел. (0382) 72-83-85, (0382) 72-85-98, (067) 384-17-50

Говорущенко Тетяна Олександрівна - кандидат технічних наук, доцент кафедри системного програмування, Хмельницький національний університет, м.Хмельницький, Україна, e-mail: **tat_yana@ukr.net**, **kism@beta.tup.km.ua**, тел. (0382) 72-83-85, (0382) 78-56-82, (095) 11-22-544