УДК 004.891.3: 004.3

МЕТОД ОЦІНЮВАННЯ ТРУДОМІСТКОСТІ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

О.П.КИКОТЬ, С.В.КОЗАК, Д.О.ДЕНИСЮК
(Науковий керівник – Т.О.Говорущенко)
Хмельницький національний університет

У статті запропоновано метод оцінювання трудомісткості розроблення програмного забезпечення на основі функційних точок, який дає можливість визначити кількість функційних точок для програмного проекту, а також дозволяє на ранніх етапах життєвого циклу оцінити розмір програмного проекту (LOC-оцінка) при його потенційній реалізації різними мовами програмування. Розроблений метод усуває залежність оцінки від суб'єктів, залучених до процесу оцінювання.

In the article the method of evaluation of the software development laboriousness based on the function points is proposed. This method provides the determination of the number of functional points for the software project, and provides early evaluation of the software project size (LOC) on its potential implementation in the different programming languages. This method eliminates the dependency of assessment from subjects involved in the evaluation process.

Ключові слова: програмне забезпечення (ПЗ), програмний проект, трудомісткість розроблення ПЗ, LOC-оцінка, функційна точка, метод функційних точок.

Вступ. Оцінювання трудомісткості розроблення ПЗ є одним з найбільш важливих видів діяльності в процесі створення ПЗ. За відсутності адекватної та достовірної оцінки неможливо забезпечити чітке планування та управління проектом. Недооцінка вартості, тривалості та ресурсів, необхідних для створення ПЗ, тягне за собою недостатню чисельність проектної команди, надмірно стислі терміни розроблення і, як результат, втрату довіри до розробників у випадку порушення графіку [1].

Моделі та методи оцінювання трудомісткості використовуються для вирішення багатьох задач, серед яких можна виділити наступні [1, 2]:

- розроблення бюджету проекту (необхідна точність);
- аналіз ступеня ризику та вибір компромісного рішення (уточнюються масштаби проекту, можливість повторного використання, кількість розробників, використовуване обладнання);
- планування та управління проектом (одержані результати забезпечують розподіл та класифікацію витрат за компонентами, етапами та операціями);
- аналіз витрат на покращення якості ПЗ (дозволяє оцінити витрати та прибуток від стратегії інвестування у вдосконалення технологій та можливості повторного використання).

Процедура оцінки трудомісткості розроблення ПЗ складається з наступних дій [1]: 1) оцінка розміру розроблюваного продукту; 2) оцінка трудомісткості (в людино-місяцях або людино-годинах); 3) оцінка тривалості проекту (в календарних місяцях); 4) оцінка вартості проекту.

Найбільш поширеними методиками оцінювання трудомісткості розроблення ПЗ є конструктивна модель вартості COCOMO, розроблена Барі Боемом, та модернізована модель COCOMO II [3-5]. Головною особливістю цих моделей є те, що для оцінювання трудомісткості необхідною є оцінка розміру розроблюваного програмного продукту.

При оцінюванні трудомісткості розроблення ПЗ наразі існують наступні проблеми [1, 2]:

- залежність оцінки від суб'єктів, залучених до процесу оцінювання;
- відсутність бази оцінок типових проектів (особливо для принципово нових програмних проектів) та систем підтримки прийняття рішень (СППР) для оцінювання трудомісткості розроблення ПЗ (хоча накопичення великої кількості оцінок для проектів різних типів значно спростило б задачу оцінювання трудомісткості);
- використання LOC-оцінок (для готового програмного коду) як найбільш поширеної одиниці вимірювання розміру ПЗ при великій кількості недоліків такої оцінки: неможливість використання на ранніх етапах життєвого циклу ПЗ, коли відсутній готовий програмний код; залежність LOC-оцінки від середовища розроблення та методів проектування; неврахування витрат, не пов'язаних із програмним кодом; неврахування обсягу автоматично генерованого та створеного програмістом коду; при використанні LOC-оцінок як основної одиниці вимірювання трудомісткість розроблення ПЗ розраховується лише для вже готового програмного коду.
- поодиноке використання оцінювання трудомісткості розроблення ПЗ на основі функційних точок (в основному через незабезпеченість такого процесу оцінювання математичним апаратом), але саме функційні точки як основна одиниця вимірювання дозволяють розраховувати трудомісткість на ранніх етапах життєвого циклу та прогнозувати LOC-оцінки ПЗ для реалізації різними мовами програмування, тобто підтримують обгрунтований вибір мови програмування для реалізації.

Враховуючи вищевикладене, *актуальною задачею* наразі є побудова математичного апарату для підтримки процесу оцінювання трудомісткості ПЗ на основі функційних точок, завдяки якому саме функційні точки можуть стати основною одиницею оцінювання трудомісткості, що дасть можливість оцінювати трудомісткість

розроблення ПЗ та прогнозувати його LOC-оцінки на ранніх етапах життєвого циклу, коли ще відсутній готовий програмний код, а також усунути залежність оцінки від суб'єктів, залучених до процесу оцінювання.

Постановка задачі. З результатів аналізу сучасного стану галузі слідує, що перспективним напрямком дослідження є побудова методу оцінювання трудомісткості розроблення ПЗ на основі функційних точок.

1. Метод оцінювання трудомісткості розроблення ПЗ на основі функційних точок. Розмір ПЗ найкраще оцінювати в термінах кількості та складності функцій, реалізованих програмним кодом, а не за допомогою кількості рядків коду [1]. *Функційна точка* – це одиниця вимірювання функційності ПЗ, що є комбінацією таких властивостей ПЗ, як: інтенсивність використання введення та виведення зовнішніх даних, взаємодія системи з користувачем, зовнішніх інтерфейсів та файлів, використовуваних програмним забезпеченням [2]. При використанні функційних точок вимірюванню підлягають категорії бізнес-функцій користувача. У [2] наведено методику (спосіб) оцінювання трудомісткості розроблення ПЗ на основі функційних точок у вигляді рекомендацій, наведено також необхідні для оцінювання категорії та вагові коефіцієнти, але відсутній формалізований метод, що призводить до вільного використання та трактування наведеної методики, тому оцінки трудомісткості і залежать від суб'єктів, залучених до процесу оцінювання.

Метод оцінювання трудомісткості розроблення ПЗ на основі функційних точок складається з наступних етапів – рис.1.

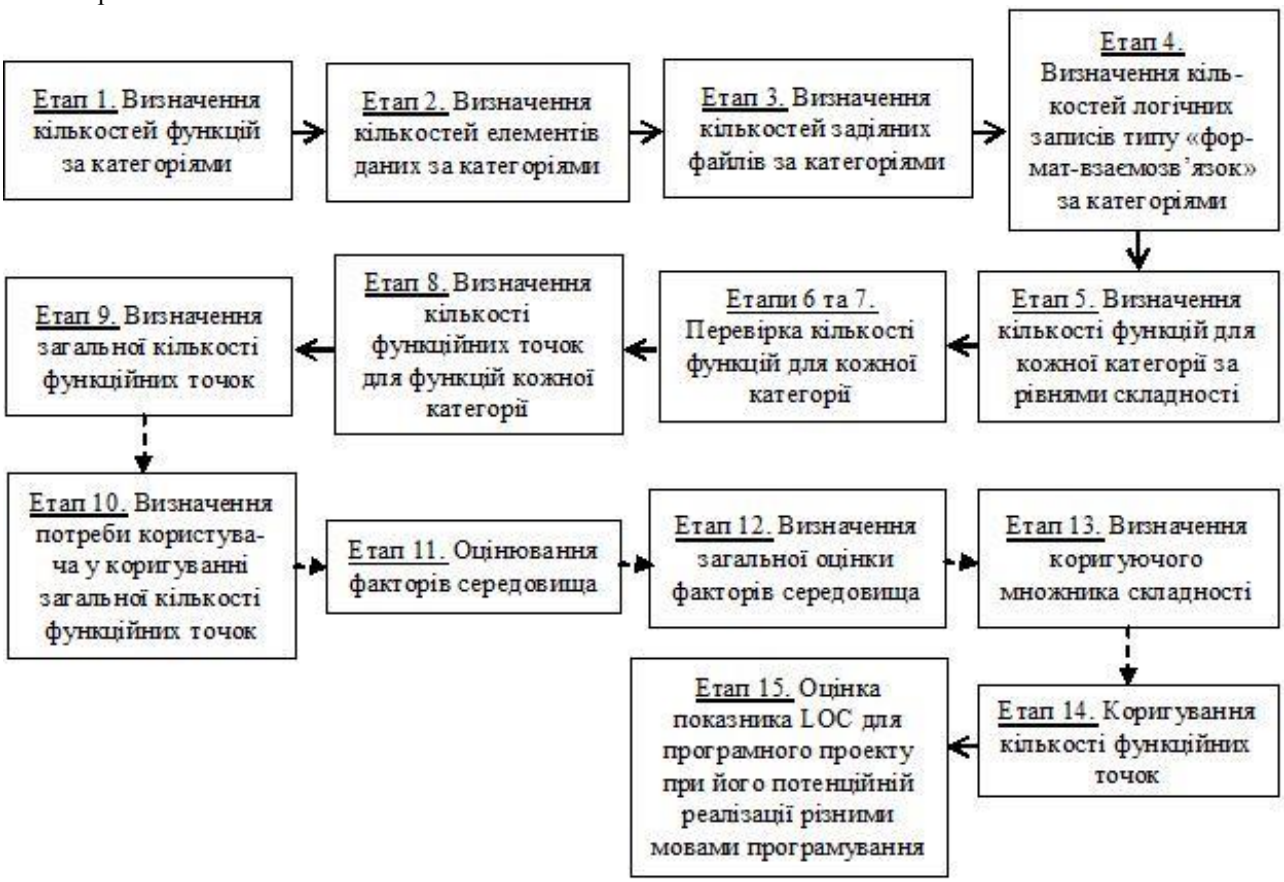

Рис. 1 Етапи методу оцінювання трудомісткості розроблення ПЗ на основі функційних точок

Деталізуємо та формалізуємо етапи методу оцінювання трудомісткості розроблення ПЗ на основі функційних точок:

1) визначення кількостей функцій за категоріями:

q_{if} - кількість функцій введення (функції введення від кінцевого користувача);

q_{of} - кількість функцій виведення (функції виведення, призначені для кінцевого користувача);

q_{cif} - кількість запитів введення (запити введення – це зовнішні специфічні команди чи запити, які виконуються ПЗ та генеруються ззовні; саме запити забезпечують безпосередній доступ до бази даних);

q_{cof} - кількість запитів виведення (запити виведення – це зовнішні специфічні команди чи запити, які виконуються ПЗ та генеруються ззовні; саме запити забезпечують безпосередній доступ до бази даних);

q_{fl} - кількість файлів (структур даних, що представляють собою первинну логічну групу даних користувача, які постійно знаходяться повністю всередині програмної системи та доступні для користувачів за допомогою введення, виведення, запитів або інтерфейсів);

q_i - кількість інтерфейсів (всі потоки даних, в тому числі й ті, які зберігаються за межами програмної системи). Для можливості застосування на ранніх етапах життєвого циклу зазначені кількості функцій за категоріями визначатимемо на основі специфікації вимог до ПЗ;

2) визначення кількостей елементів даних за категоріями: $QDE_{if} = \{qde_{if_1},...,qde_{if_{q_{if}}}\}$ - множина кількостей елементів даних для функцій введення; $QDE_{of} = \{qde_{of_1},...,qde_{of_{q_{of}}}\}$ - множина кількостей елементів даних для функцій виведення; $QDE_{cif} = \{qde_{cif_1},...,qde_{cif_{q_{cif}}}\}$ - множина кількостей елементів даних для запитів введення; $QDE_{cof} = \{qde_{cof_1},...,qde_{cof_{q_{cof}}}\}$ - множина кількостей елементів даних для запитів виведення; $QDE_{fl} = \{qde_{fl_1},...,qde_{fl_{q_{fl}}}\}$ - множина кількостей елементів даних для файлів; $QDE_i = \{qde_{i1},...,qde_{iq_i}\}$ - множина кількостей елементів даних для інтерфейсів;

3) визначення кількостей задіяних файлів за категоріями: $QF_{if} = \{qf_{if_1},...,qf_{if_{q_{if}}}\}$ - множина кількостей задіяних файлів для функцій введення; $QF_{of} = \{qf_{of_1},...,qf_{of_{q_{of}}}\}$ - множина кількостей задіяних файлів для функцій виведення; $QF_{cif} = \{qf_{cif_1},...,qf_{cif_{q_{cif}}}\}$ - множина кількостей задіяних файлів для запитів введення; $QF_{cof} = \{qf_{cof_1},...,qf_{cof_{q_{cof}}}\}$ - множина кількостей задіяних файлів для запитів виведення;

4) визначення кількостей логічних записів типу «формат-взаємозв'язок» за категоріями: $QLR_{fl} = \{qlr_{fl_1},...,qlr_{fl_{q_{fl}}}\}$ - множина кількостей логічних записів для файлів; $QLR_i = \{qlr_{i1},...,qlr_{iq_i}\}$ - множина кількостей логічних записів для інтерфейсів;

5) визначення кількості функцій для кожної категорії за рівнями складності (існує три рівні складності: sp - простий, md - середній, cp - складний): $q_{if_{sp}}$ - кількість простих функцій введення; $q_{if_{md}}$ - кількість функцій введення середнього рівня складності; $q_{if_{cp}}$ - кількість складних функцій введення; $q_{of_{sp}}$ - кількість простих функцій виведення; $q_{of_{md}}$ - кількість функцій виведення середнього рівня складності; $q_{of_{cp}}$ - кількість складних функцій виведення; $q_{cif_{sp}}$ - кількість простих запитів введення; $q_{cif_{md}}$ - кількість запитів введення середнього рівня складності; $q_{cif_{cp}}$ - кількість складних запитів введення; $q_{cof_{sp}}$ - кількість простих запитів виведення; $q_{cof_{md}}$ - кількість запитів виведення середнього рівня складності; $q_{cof_{cp}}$ - кількість складних запитів виведення; $q_{fl_{sp}}$ - кількість простих файлів; $q_{fl_{md}}$ - кількість файлів середнього рівня складності; $q_{fl_{cp}}$ - кількість складних файлів; $q_{i_{sp}}$ - кількість простих інтерфейсів; $q_{i_{md}}$ - кількість інтерфейсів середнього рівня складності; $q_{i_{cp}}$ - кількість складних інтерфейсів.

В усі вищевказані змінні записуємо значення «0» на початку процесу підрахунку: $q_{if_{sp}} = 0$; $q_{if_{md}} = 0$; $q_{if_{cp}} = 0$; $q_{of_{sp}} = 0$; $q_{of_{md}} = 0$; $q_{of_{cp}} = 0$; $q_{cif_{sp}} = 0$; $q_{cif_{md}} = 0$; $q_{cif_{cp}} = 0$; $q_{cof_{sp}} = 0$; $q_{cof_{md}} = 0$; $q_{cof_{cp}} = 0$; $q_{fl_{sp}} = 0$; $q_{fl_{md}} = 0$; $q_{fl_{cp}} = 0$; $q_{i_{sp}} = 0$; $q_{i_{md}} = 0$; $q_{i_{cp}} = 0$.

Визначення кількості функцій для кожної категорії кожного рівня складності відбувається на основі наступних правил (числові константи для побудови цих правил рекомендовані у [2]):

- для функцій введення ($i = \overline{1..q_{if}}$):
  1. $q_{if_{sp}} = q_{if_{sp}} + 1 \quad \forall(qde_{if_i} \in \{1,...,19\}) \wedge (qf_{if_i} \in \{0,1\})$
  2. $q_{if_{sp}} = q_{if_{sp}} + 1 \quad \forall(qde_{if_i} \in \{1,...,5\}) \wedge (qf_{if_i} \in \{2,3\})$

3. $q_{if_{md}} = q_{if_{md}} + 1 \;\; \forall (qde_{if_i} \in \{1,...,5\}) \wedge (qf_{if_i} \geq 4)$
4. $q_{if_{md}} = q_{if_{md}} + 1 \;\; \forall (qde_{if_i} \in \{6,...,19\}) \wedge (qf_{if_i} \in \{2,3\})$
5. $q_{if_{md}} = q_{if_{md}} + 1 \;\; \forall (qde_{if_i} \geq 20) \wedge (qf_{if_i} \in \{0,1\})$
6. $q_{if_{cp}} = q_{if_{cp}} + 1 \;\; \forall (qde_{if_i} \geq 20) \wedge (qf_{if_i} \in \{2,3\})$
7. $q_{if_{cp}} = q_{if_{cp}} + 1 \;\; \forall (qde_{if_i} \geq 6) \wedge (qf_{if_i} \geq 4)$

- для функцій виведення ($j = \overline{1..q_{of}}$):

8. $q_{of_{sp}} = q_{of_{sp}} + 1 \;\; \forall (qde_{of_j} \in \{1,...,19\}) \wedge (qf_{of_j} \in \{0,1\})$
9. $q_{of_{sp}} = q_{of_{sp}} + 1 \;\; \forall (qde_{of_j} \in \{1,...,5\}) \wedge (qf_{of_j} \in \{2,3\})$
10. $q_{of_{md}} = q_{of_{md}} + 1 \;\; \forall (qde_{of_j} \in \{1,...,5\}) \wedge (qf_{of_j} \geq 4)$
11. $q_{of_{md}} = q_{of_{md}} + 1 \;\; \forall (qde_{of_j} \in \{6,...,19\}) \wedge (qf_{of_j} \in \{2,3\})$
12. $q_{of_{md}} = q_{of_{md}} + 1 \;\; \forall (qde_{of_j} \geq 20) \wedge (qf_{of_j} \in \{0,1\})$
13. $q_{of_{cp}} = q_{of_{cp}} + 1 \;\; \forall (qde_{of_j} \geq 20) \wedge (qf_{of_j} \in \{2,3\})$
14. $q_{of_{cp}} = q_{of_{cp}} + 1 \;\; \forall (qde_{of_j} \geq 6) \wedge (qf_{of_j} \geq 4)$

- для запитів введення ($k = \overline{1..q_{cif}}$):

15. $q_{cif_{sp}} = q_{cif_{sp}} + 1 \;\; \forall (qde_{cif_k} \in \{1,...,19\}) \wedge (qf_{cif_k} \in \{0,1\})$
16. $q_{cif_{sp}} = q_{cif_{sp}} + 1 \;\; \forall (qde_{cif_k} \in \{1,...,5\}) \wedge (qf_{cif_k} \in \{2,3\})$
17. $q_{cif_{md}} = q_{cif_{md}} + 1 \;\; \forall (qde_{cif_k} \in \{1,...,5\}) \wedge (qf_{cif_k} \geq 4)$
18. $q_{cif_{md}} = q_{cif_{md}} + 1 \;\; \forall (qde_{cif_k} \in \{6,...,19\}) \wedge (qf_{cif_k} \in \{2,3\})$
19. $q_{cif_{md}} = q_{cif_{md}} + 1 \;\; \forall (qde_{cif_k} \geq 20) \wedge (qf_{cif_k} \in \{0,1\})$
20. $q_{cif_{cp}} = q_{cif_{cp}} + 1 \;\; \forall (qde_{cif_k} \geq 20) \wedge (qf_{cif_k} \in \{2,3\})$
21. $q_{cif_{cp}} = q_{cif_{cp}} + 1 \;\; \forall (qde_{cif_k} \geq 6) \wedge (qf_{cif_k} \geq 4)$

- для запитів виведення ($g = \overline{1..q_{cof}}$):

22. $q_{cof_{sp}} = q_{cof_{sp}} + 1 \;\; \forall (qde_{cof_g} \in \{1,...,19\}) \wedge (qf_{cof_g} \in \{0,1\})$
23. $q_{cof_{sp}} = q_{cof_{sp}} + 1 \;\; \forall (qde_{cof_g} \in \{1,...,5\}) \wedge (qf_{cof_g} \in \{2,3\})$
24. $q_{cof_{md}} = q_{cof_{md}} + 1 \;\; \forall (qde_{cof_g} \in \{1,...,5\}) \wedge (qf_{cof_g} \geq 4)$
25. $q_{cof_{md}} = q_{cof_{md}} + 1 \;\; \forall (qde_{cof_g} \in \{6,...,19\}) \wedge (qf_{cof_g} \in \{2,3\})$
26. $q_{cof_{md}} = q_{cof_{md}} + 1 \;\; \forall (qde_{cof_g} \geq 20) \wedge (qf_{cof_g} \in \{0,1\})$
27. $q_{cof_{cp}} = q_{cof_{cp}} + 1 \;\; \forall (qde_{cof_g} \geq 20) \wedge (qf_{cof_g} \in \{2,3\})$
28. $q_{cof_{cp}} = q_{cof_{cp}} + 1 \;\; \forall (qde_{cof_g} \geq 6) \wedge (qf_{cof_g} \geq 4)$

- для файлів ($n = \overline{1..q_{fl}}$):

29. $q_{fl_{sp}} = q_{fl_{sp}} + 1 \;\; \forall (qde_{fl_n} \in \{1,...,50\}) \wedge (qlr_{fl_n} = 1)$
30. $q_{fl_{sp}} = q_{fl_{sp}} + 1 \;\; \forall (qde_{fl_n} \in \{1,...,19\}) \wedge (qlr_{fl_n} \in \{2,...,5\})$
31. $q_{fl_{md}} = q_{fl_{md}} + 1 \;\; \forall (qde_{fl_n} \in \{1,...,19\}) \wedge (qlr_{fl_n} \geq 6)$
32. $q_{fl_{md}} = q_{fl_{md}} + 1 \;\; \forall (qde_{fl_n} \in \{20,...,50\}) \wedge (qlr_{fl_n} \in \{2,...,5\})$

33. $q_{fl_{md}} = q_{fl_{md}} + 1 \ \ \forall (qde_{fl_n} \geq 51) \wedge (qlr_{fl_n} = 1)$
34. $q_{fl_{cp}} = q_{fl_{cp}} + 1 \ \ \forall (qde_{fl_n} \geq 51) \wedge (qlr_{fl_n} \in \{2,...,5\})$
35. $q_{fl_{cp}} = q_{fl_{cp}} + 1 \ \ \forall (qde_{fl_n} \geq 20) \wedge (qlr_{fl_n} \geq 6)$

- для інтерфейсів ($l = \overline{1..q_i}$):

36. $q_{i_{sp}} = q_{i_{sp}} + 1 \ \ \forall (qde_{i_l} \in \{1,...,50\}) \wedge (qlr_{i_l} = 1)$
37. $q_{i_{sp}} = q_{i_{sp}} + 1 \ \ \forall (qde_{i_l} \in \{1,...,19\}) \wedge (qlr_{i_l} \in \{2,...,5\})$
38. $q_{i_{md}} = q_{i_{md}} + 1 \ \ \forall (qde_{i_l} \in \{1,...,19\}) \wedge (qlr_{i_l} \geq 6)$
39. $q_{i_{md}} = q_{i_{md}} + 1 \ \ \forall (qde_{i_l} \in \{20,...,50\}) \wedge (qlr_{i_l} \in \{2,...,5\})$
40. $q_{i_{md}} = q_{i_{md}} + 1 \ \ \forall (qde_{i_l} \geq 51) \wedge (qlr_{i_l} = 1)$
41. $q_{i_{cp}} = q_{i_{cp}} + 1 \ \ \forall (qde_{i_l} \geq 51) \wedge (qlr_{i_l} \in \{2,...,5\})$
42. $q_{i_{cp}} = q_{i_{cp}} + 1 \ \ \forall (qde_{i_l} \geq 20) \wedge (qlr_{i_l} \geq 6)$

6) перевірка кількості функцій в кожній категорії за наступними правилами:

- якщо $q_{if} \neq q_{if_{sp}} + q_{if_{md}} + q_{if_{cp}}$, то відбулась помилка при підрахунку кількості функцій введення за рівнями складності ($a = a + 1$);
- якщо $q_{of} \neq q_{of_{sp}} + q_{of_{md}} + q_{of_{cp}}$, то відбулась помилка при підрахунку кількості функцій виведення за рівнями складності ($a = a + 1$);
- якщо $q_{cif} \neq q_{cif_{sp}} + q_{cif_{md}} + q_{cif_{cp}}$, то відбулась помилка при підрахунку кількості запитів введення за рівнями складності ($a = a + 1$);
- якщо $q_{cof} \neq q_{cof_{sp}} + q_{cof_{md}} + q_{cof_{cp}}$, то відбулась помилка при підрахунку кількості запитів виведення за рівнями складності ($a = a + 1$);
- якщо $q_{fl} \neq q_{fl_{sp}} + q_{fl_{md}} + q_{fl_{cp}}$, то відбулась помилка при підрахунку кількості файлів за рівнями складності ($a = a + 1$);
- якщо $q_i \neq q_{i_{sp}} + q_{i_{md}} + q_{i_{cp}}$, то відбулась помилка при підрахунку кількості інтерфейсів за рівнями складності ($a = a + 1$);

7) якщо $a \neq 0$, то повернутись до кроку 5, інакше, якщо $a = 0$, перейти до кроку 8;

8) визначення кількості функційних точок для функцій кожної категорії за наступними формулами (із застосуванням вагових коефіцієнтів складності, наведених у [2]):

- для функцій введення: $q_{if_{FP}} = 3 \cdot q_{if_{sp}} + 4 \cdot q_{if_{md}} + 6 \cdot q_{if_{cp}}$;
- для функцій виведення: $q_{of_{FP}} = 4 \cdot q_{of_{sp}} + 5 \cdot q_{of_{md}} + 7 \cdot q_{of_{cp}}$;
- для запитів введення: $q_{cif_{FP}} = 3 \cdot q_{cif_{sp}} + 4 \cdot q_{cif_{md}} + 6 \cdot q_{cif_{cp}}$;
- для запитів виведення: $q_{cof_{FP}} = 4 \cdot q_{cof_{sp}} + 5 \cdot q_{cof_{md}} + 7 \cdot q_{cof_{cp}}$;
- для файлів: $q_{fl_{FP}} = 7 \cdot q_{fl_{sp}} + 10 \cdot q_{fl_{md}} + 15 \cdot q_{fl_{cp}}$;
- для інтерфейсів: $q_{iFP} = 5 \cdot q_{i_{sp}} + 7 \cdot q_{i_{md}} + 10 \cdot q_{i_{cp}}$;

9) визначення загальної кількості функційних точок:

$q_{FP} = q_{if_{FP}} + q_{of_{FP}} + q_{cif_{FP}} + q_{cof_{FP}} + q_{fl_{FP}} + q_{iFP}$

10) визначення потреби користувача у коригуванні загальної кількості функційних точок з врахуванням факторів середовища, які впливають на процес розроблення ПЗ в цілому: якщо таке коригування необхідне, то виконуються етапи 11-14, інакше відбувається перехід до етапу 15;

11) оцінювання факторів середовища (детальний опис факторів наведено у [2]): ef_{dtc} - оцінка каналів передачі даних; ef_{dc} - оцінка розподілених обчислень; ef_{pr} - оцінка вимог до продуктивності; ef_{cr} - оцінка конфігурування з обмеженнями; ef_{tf} - оцінка частоти транзакцій; ef_{ir} - оцінка інтерактивного запиту; ef_{ue} -

оцінка ефективності на рівні кінцевого користувача; ef_{iu} - оцінка інтерактивного оновлення; ef_{cp} - оцінка складного опрацювання; ef_{ru} - оцінка повторного використання; ef_{fci} - оцінка спрощення перетворення/інсталяції; ef_{so} - оцінка спрощення операції; ef_{umn} - оцінка використання на декількох вузлах; ef_{pcf} - оцінка потенціалу зміни функції. Оцінювання відбувається за шкалою від 0 до 5, де значення 0 означає неможливість застосування фактору [2].

12) визначення загальної оцінки факторів середовища:

$$ef = ef_{dtc} + ef_{dc} + ef_{pr} + ef_{cr} + ef_{tf} + ef_{ir} + ef_{ue} + ef_{iu} + ef_{cp} + ef_{ru} + ef_{fci} + ef_{so} + ef_{umn} + ef_{pcf}$$

13) визначення коригуючого множника складності [2]:

$$caf = 0{,}65 + (0{,}01 \cdot ef)$$

14) коригування кількості функційних точок:

$$q_{FP_{correct}} = q_{FP} \cdot caf$$

15) прогнозована оцінка показника LOC для даного програмного проекту при його потенційній реалізації різними мовами програмування (середні показники LOC для різних мов програмування з розрахунку на одну функційну точку наведені у [2]):

$$LOC_{lang} = q_{FP_{correct}} \cdot LOC_{lang_{av}},$$

де $LOC_{lang_{av}}$ - середній показник LOC для мови програмування $lang$, взятий з [2]; LOC_{lang} - оцінка показника LOC для даного програмного проекту мовою $lang$; $q_{FP_{correct}}$ замінюється показником q_{FP}, якщо корекція не проводилась.

Розроблений метод оцінювання трудомісткості на основі функційних точок відрізняється від існуючих методів тим, що є формалізованим, за рахунок чого усуває залежність оцінки від суб'єктів, залучених до процесу оцінювання. Переваги розробленого методу: можливість його використання на ранніх етапах життєвого циклу ПЗ; відсутність залежності від використовуваної мови програмування, методології та технології; можливість вимірювання LOC-оцінок для реалізації проекту різними мовами програмування; можливість застосування даного методу для програмних систем із графічним інтерфейсом користувача; врахування факторів середовища за потреби.

2. Експериментальна частина. Наведемо приклад застосування розробленого методу оцінювання трудомісткості на основі функційних точок. Розглянемо специфікацію вимог до програмного проекту Olive Insurance Client-Management System [6].

Для цього проекту кількості функцій за категоріями складають: $q_{if} = 20$; $q_{of} = 25$; $q_{cif} = 12$ $q_{cof} = 14$; $q_{fl} = 14$; $q_i = 15$; кількості елементів даних за категоріями складають:

$QDE_{if} = \{1,5,3,2,4,6,8,1,10,11,21,23,15,23,7,9,22,3,5,7\}$;

$QDE_{of} = \{2,3,5,7,9,14,3,4,18,16,23,22,1,2,3,8,19,25,21,4,8,9,13,17,26\}$;

$QDE_{cif} = \{6,8,10,19,21,1,3,7,8,4,23,24\}$;

$QDE_{cof} = \{21,23,24,1,3,7,12,6,13,18,19,2,5,21\}$;

$QDE_{fl} = \{23,38,41,2,4,8,7,5,4,2,51,53,1,56\}$; $QDE_i = \{34,32,67,6,5,8,23,25,53,3,7,8,12,13,17\}$;

кількості задіяних файлів за категоріями складають:

$QF_{if} = \{2,3,1,4,5,1,2,3,4,5,4,5,3,2,1,3,2,4,6,1\}$;

$QF_{of} = \{4,3,5,6,1,2,3,4,5,3,5,4,4,4,2,1,3,2,3,4,5,3,2,1,4\}$;

$QF_{cif} = \{0,2,3,1,4,5,2,1,2,2,1,1\}$;

$QF_{cof} = \{1,2,1,0,0,3,4,2,1,3,2,1,6,5\}$;

кількості логічних записів типу «формат-взаємозв'язок» за категоріями складають:

$QLR_{fl} = \{1,2,5,4,6,7,4,3,2,1,2,4,1,8\}$;

$QLR_i = \{1,6,7,5,4,7,8,4,3,4,5,6,2,1,5\}$.

Визначимо кількості функцій кожної категорії кожного рівня складності на основі правил етапу 5. Наприклад, детально розглянемо процес визначення кількостей функцій різних рівнів складності для функцій введення за правилами 1-7: $(qde_{if_1} \in \{1,...,5\}) \wedge (qf_{if_1} \in \{2,3\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 2;

$(qde_{if_2} \in \{1,...,5\}) \wedge (qf_{if_2} \in \{2,3\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 2; $(qde_{if_3} \in \{1,...,19\}) \wedge (qf_{if_3} \in \{0,1\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 1; $(qde_{if_4} \in \{1,...,5\}) \wedge (qf_{if_4} \geq 4)$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 3; $(qde_{if_5} \in \{1,...,5\}) \wedge (qf_{if_5} \geq 4)$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 3; $(qde_{if_6} \in \{1,...,19\}) \wedge (qf_{if_6} \in \{0,1\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 1; $(qde_{if_7} \in \{6,...,19\}) \wedge (qf_{if_7} \in \{2,3\})$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 4; $(qde_{if_8} \in \{1,...,5\}) \wedge (qf_{if_8} \in \{2,3\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 2; $(qde_{if_9} \geq 6 \wedge (qf_{if_9} \geq 4)$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 7; $(qde_{if_{10}} \geq 6) \wedge (qf_{if_{10}} \geq 4)$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 7; $(qde_{if_{11}} \geq 6) \wedge (qf_{if_{11}} \geq 4)$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 7; $(qde_{if_{12}} \geq 6) \wedge (qf_{if_{12}} \geq 4)$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 7; $(qde_{if_{13}} \in \{6,...,19\}) \wedge (qf_{if_{13}} \in \{2,3\})$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 4; $(qde_{if_{14}} \geq 20) \wedge (qf_{if_{14}} \in \{2,3\})$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 6; $(qde_{if_{15}} \in \{1,...,19\}) \wedge (qf_{if_{15}} \in \{0,1\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 1; $(qde_{if_{16}} \in \{6,...,19\}) \wedge (qf_{if_{16}} \in \{2,3\})$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 4; $(qde_{if_{17}} \geq 20) \wedge (qf_{if_{17}} \in \{2,3\})$, тому $q_{if_{cp}} = q_{if_{cp}} + 1$ - за правилом 6; $(qde_{if_{18}} \in \{1,...,5\}) \wedge (qf_{if_{18}} \geq 4)$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 3; $qde_{if_{19}} \in \{1,...,5\}) \wedge (qf_{if_{19}} \geq 4)$, тому $q_{if_{md}} = q_{if_{md}} + 1$ - за правилом 3; $qde_{if_{20}} \in \{1,...,19\}) \wedge (qf_{if_{20}} \in \{0,1\})$, тому $q_{if_{sp}} = q_{if_{sp}} + 1$ - за правилом 1.

Тоді для функцій введення мають місце наступні кількості функцій за рівнями складності:

$q_{if_{sp}} = 7$; $q_{if_{md}} = 7$; $q_{if_{cp}} = 6$.

Аналогічно обчислимо кількості функцій інших категорій за рівнями складності:

- за правилами 8-14: $q_{of_{sp}} = 6$; $q_{of_{md}} = 11$; $q_{of_{cp}} = 8$;
- за правилами 15-21: $q_{cif_{sp}} = 5$; $q_{cif_{md}} = 6$; $q_{cif_{cp}} = 1$;
- за правилами 22-28: $q_{cof_{sp}} = 4$; $q_{cof_{md}} = 7$; $q_{cof_{cp}} = 3$;
- за правилами 29-35: $q_{fl_{sp}} = 7$; $q_{fl_{md}} = 4$; $q_{fl_{cp}} = 3$;
- за правилами 36-42: $q_{i_{sp}} = 8$; $q_{i_{md}} = 3$; $q_{i_{cp}} = 4$.

Перевірка кількості функцій в кожній категорії показала, що помилки при підрахунку кількостей функцій різних категорій за рівнями складності відсутні, тоді перейдемо до етапу 8 і визначимо кількості функційних точок для функцій кожної категорії:

- для функцій введення: $q_{if_{FP}} = 3 \cdot 7 + 4 \cdot 7 + 6 \cdot 6 = 85$;
- для функцій виведення: $q_{of_{FP}} = 4 \cdot 6 + 5 \cdot 11 + 7 \cdot 8 = 135$;
- для запитів введення: $q_{cif_{FP}} = 3 \cdot 5 + 4 \cdot 6 + 6 \cdot 1 = 45$;
- для запитів виведення: $q_{cof_{FP}} = 4 \cdot 4 + 5 \cdot 7 + 7 \cdot 3 = 72$;
- для файлів: $q_{fl_{FP}} = 7 \cdot 7 + 10 \cdot 4 + 15 \cdot 3 = 134$;
- для інтерфейсів: $q_{i_{FP}} = 5 \cdot 8 + 7 \cdot 3 + 10 \cdot 4 = 101$.

Тоді загальна кількість функційних точок складає:

$q_{FP} = 85 + 135 + 45 + 72 + 134 + 101 = 572$.

Для даного проекту відсутня потреба у коригуванні загальної кількості функційних точок з врахуванням факторів середовища, тому оцінимо показник LOC для даного програмного проекту при його реалізації мовами програмування:

- C++: $LOC_{C++} = 572 \cdot 53 = 30316$;
- Delphi: $LOC_{Delphi} = 572 \cdot 29 = 16588$;

- Java: $LOC_{Java} = 572 \cdot 53 = 30316.$

Результати експерименту показали, що розглянутий проект має трудомісткість розроблення у 572 функційних точки. Реалізація даного програмного проекту мовою Delphi матиме найменшу кількість рядків програмного коду (прогнозовано 16588 рядків) в порівнянні з двома іншими обраними мовами програмування.

Для емпіричної оцінки достовірності розробленого методу використаємо порівняння отриманих результатів експерименту з LOC-оцінкою коду, розробленого за розглянутою специфікацією проекту Olive Insurance Client-Management System. В результаті розроблення програмного коду мовою C++ для зазначеного проекту програмісти отримали $LOC_{C++real} = 30528$ рядків коду, тоді емпірична оцінка достовірності прогнозованої LOC-оцінки складає:

$$D_{C++} = \frac{LOC_{C++}}{LOC_{C++_{real}}} = \frac{30316}{30528} = 0{,}993;$$

похибка прогнозованої LOC-оцінки складає:

$$\varepsilon = \frac{\left|LOC_{C++} - LOC_{C++_{real}}\right|}{LOC_{C++}} = \frac{\left|30316 - 30528\right|}{30316} = 0{,}007.$$

Отже, емпірична оцінка достовірності та похибки прогнозованої LOC-оцінки доводить високу достовірність (більше 99%) роботи запропонованого методу оцінювання трудомісткості розроблення програмного забезпечення.

Висновки. Запропоновано метод оцінювання трудомісткості розроблення ПЗ на основі функційних точок, який дає можливість визначити кількість функційних точок для програмного проекту, а також дозволяє на ранніх етапах життєвого циклу оцінити розмір програмного проекту (LOC-оцінка) при його потенційній реалізації різними мовами програмування.

Представлений метод забезпечує процес оцінювання трудомісткості розроблення ПЗ на основі функційних точок математичним підґрунтям, внаслідок чого усуває залежність оцінки від суб'єктів, залучених до процесу оцінювання, а також вирішує проблему поодинокого використання оцінювання трудомісткості розроблення ПЗ на основі функційних точок. Прогнозована LOC-оцінка, надана розробленим методом, дозволяє оцінити вартість та тривалість даного програмного проекту на ранніх етапах життєвого циклу ПЗ, а також підтримує обґрунтований вибір мови програмування завдяки можливості порівняння прогнозованих LOC-оцінок для реалізації проекту різними мовами програмування.

Результати проведеного експерименту показали, що розглянутий проект Olive Insurance Client-Management System має трудомісткість розроблення у 572 функційних точки. Реалізація даного програмного проекту мовою Delphi матиме найменшу кількість рядків програмного коду (прогнозовано 16588 рядків) в порівнянні з двома іншими обраними мовами програмування. Експериментальна частина роботи підтверджує застосування розробленого методу для оцінювання трудомісткості розроблення ПЗ з достовірністю 99,3%.

Перспективою для подальших досліджень є:

1) розроблення методу (способу, алгоритму) оцінювання факторів середовища;
2) розроблення методу оцінювання трудомісткості розроблення ПЗ на основі точок властивостей;
3) розроблення методу (способу, алгоритму) оцінювання тривалості та вартості розроблення ПЗ на основі отриманих розробленими методами LOC-оцінок (на основі моделей COCOMO та COCOMO II);
4) розроблення системи підтримки прийняття рішень для оцінювання трудомісткості розроблення ПЗ на основі функційних точок та точок властивостей, в основу якої будуть покладені математичні методи.

Перелік посилань:

1. А.М. Вендров. Проектирование программного обеспечения экономических информационных систем: Учебник. - М.: Финансы и статистика, 2006. – 544с.
2. Р.Т.Фатрелл. Управление программными проектами: достижение оптимального качества при минимуме затрат / Р.Т.Фатрелл, Д.Ф.Шафер, Л.И.Шафер – М.: Издательский дом «Вильямс», 2003. – 1136 с.
3. Э.Брауде. Технология разработки программного обеспечения - СПб:Питер,2004 -655 с.
4. R.W.Selby. Software Engineering: Barry W. Boehm's Lifetime Contributions to Software Development, Management, and Research - Wiley/IEEE Computer Society Press, 2007. - 832 p.
5. И.Соммервилл. Инженерия программного обеспечения. 6-е издание. - Издательский дом "Вильямс", 2002 – 624 с.
6. Software Requirement Specifications [Full Version] // [Electronic resource] - Access mode: http://findebookee.com/s/software-requirement-specifications

Рецензенти: Савенко О.С., к.т.н., декан ФПКТС, доц. каф.СПр ХНУ;
Кльоц Ю.П., к.т.н., доц. каф.СПр ХНУ