УДК 004.891.3: 004.3

Т.О. ГОВОРУЩЕНКО, В.М. СТЕЦЮК, Я.Л. НОВИЦЬКИЙ
Хмельницький національний університет

МЕТОД ОЦІНЮВАННЯ КІЛЬКОСТІ АВТОМАТИЧНО ГЕНЕРОВАНОГО ПРОГРАМНОГО КОДУ

Метою даного дослідження є розроблення методу оцінювання кількості автоматично генерованого програмного коду для підрахунку кількості рядків автоматично генерованого програмного коду та реальної кількості рядків коду, створеного саме програмістом, яка дасть можливість більш точно і достовірно оцінити трудомісткість розроблення програмного забезпечення на основі LOC-оцінки.

Ключові слова: програмне забезпечення (ПЗ), трудомісткість розроблення ПЗ, метод алгоритмічного моделювання, LOC-оцінка, автоматично генерований програмний код.

T.O. HOVORUSHCHENKO, V.M. STECYUK, Y.L. NOVYTSKYI
Khmelnytsky National University

METHOD OF EVALUATING THE QUANTITY OF AUTOMATICALLY GENEREATED SOURCE CODE

The aim of this study is the development of: the method of evaluating the quantity of automatically generated source code for the calculating the number of lines of automatically generated source code and the real quantity of lines of code, which created by the programmer. It will provide the more accurate and reliable evaluations of the efforts for software development on the basis of the LOC-evaluations. Software development is not always successful due to non-compliance with the planned project deadlines or budget overruns, which is often due to an incorrect assessment of the efforts for the software development. One of the main methods for evaluating the efforts for software development is algorithmic simulation - the method, in which the dependence of the efforts for the project on a quantitative indicator of software is determined. The most influential factor of evaluating the efforts in these models is the number of lines of code or LOC-evaluation. As the analysis showed, LOC-evaluation has many problems, including the distortion of LOC-evaluation, because it is impossible to separate the number of lines of code, which is automatically generated by the development environment, and the number of lines of code, which is created manually by the programmer. In this paper, the production rules and the method for evaluating the quantity of automatically generated source code are developed. These rules and method provide: evaluating the number of lines of automatically generated code; evaluating the number of using of each automatically generated element for the concrete programming language; evaluating the number of lines of code, which is written by the programmer. The prospect for further authors' research is the development of the software tool for evaluating the quantity of automatically generated source code on the basis of the developed production rules and method of evaluating the quantity of automatically generated source code. This tool will analyze the source code and will evaluate the number of lines of automatically generated code, the number of using of each automatically generated element for the concrete programming language, and the number of lines of code, which is written by the programmer.

Keywords: software, efforts for the software development, method of algorithmic simulation, LOC-evaluation, automatically generated source code.

Вступ

В процесі створення програмного забезпечення (ПЗ) одним з найбільш важливих видів діяльності є оцінка трудомісткості розроблення ПЗ [1–3]. Оцінка трудомісткості використовується для вирішення багатьох задач, серед яких можна виділити наступні [1, 4–6]:

1) розроблення бюджету програмного проекту (необхідна точність);

2) аналіз ступеня ризику та вибір компромісного рішення (уточнюються масштаби програмного проекту, можливість повторного використання, кількість розробників, використовуване обладнання);

3) планування та управління програмним проектом (одержані результати забезпечують розподіл та класифікацію витрат за компонентами, етапами та операціями);

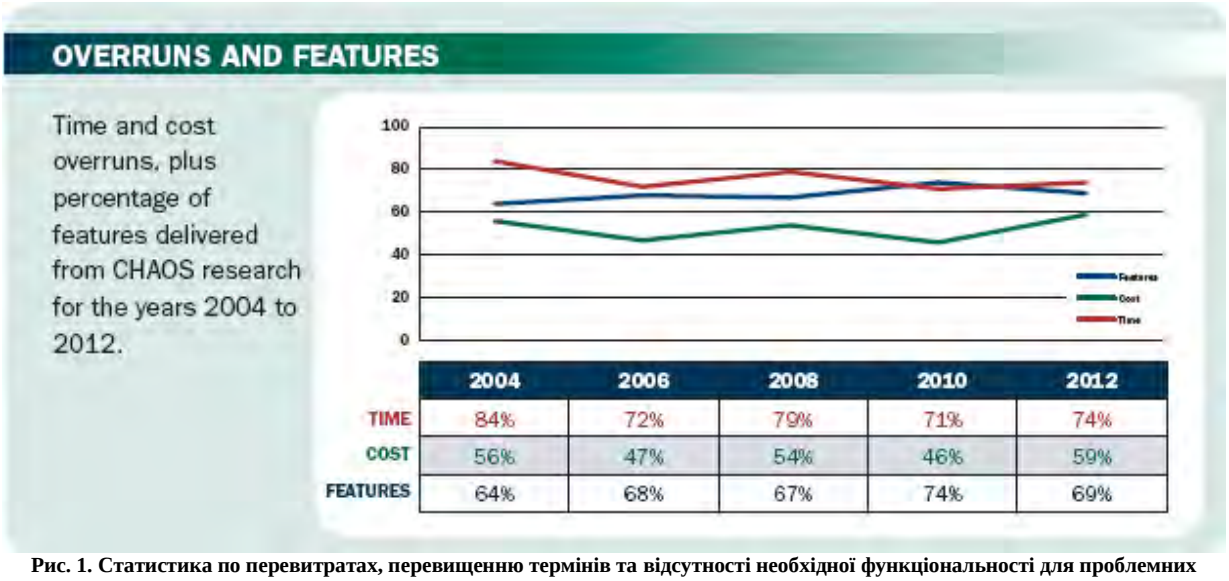
4) аналіз витрат на покращення якості ПЗ (дозволяє оцінити витрати та прибуток від стратегії інвестування у вдосконалення технологій та можливості повторного використання).

За відсутності адекватної та достовірної оцінки трудомісткості неможливо забезпечити чітке планування та управління програмним проектом. Програмні проекти, які мали перевищення термінів, перевитрати або не мали необхідних можливостей та функцій вважаються проблемними. 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 [7, 8]. Статистика по перевитратах, перевищенню термінів та відсутності необхідної функціональності для проблемних програмних проектів, за даними The Standish Group International (CHAOS reports) [7, 8], представлена на рис. 1.

Статистика [7, 8] показує також,що лише 16% програмних проектів успішно завершуються середніми компаніями у визначені термін та бюджет. Ситуація з великими компаніями значно гірша – тільки 9% проектів вкладаються у визначені терміни та бюджет. Дослідження міжнародної компанії McKinsey & Company [9] спільно з University of Oxford також показали, що половина масштабних програмних проектів з загальним бюджетом понад 15 мільйонів доларів значно перевищують заплановані витрати, зокрема: середнє перевищення витрат на проект складає 66%, середнє перевищення часу реалізації проекту – 33%, а середнє число втрат прибутку – 17% (рис. 2, 3 [9]).

З іншого боку, переоцінка при оцінюванні трудомісткості розроблення ПЗ може виявитись нічим не краще. Якщо для проекту виділено більше ресурсів, ніж реально необхідно, причому без належного контролю за їх використанням, то про жодну економію не може йти мова. Такий програмний проект

виявиться більш дорогим, ніж мав би бути при грамотній оцінці, і призведе до запізнення початку наступного проекту.



	2004	2006	2008	2010	2012
TIME	84%	72%	79%	71%	74%
COST	56%	47%	54%	46%	59%
FEATURES	64%	68%	67%	74%	69%

Рис. 1. Статистика по перевитратах, перевищенню термінів та відсутності необхідної функціональності для проблемних програмних проектів

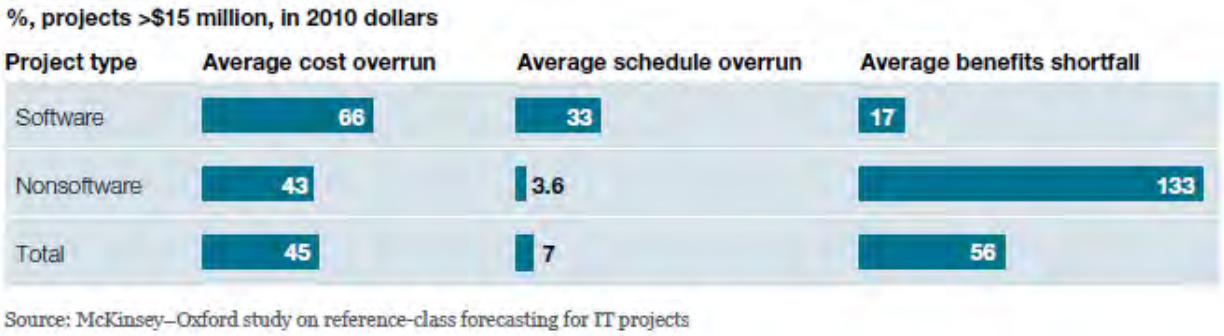


Рис. 2. Продуктивність програмних проектів різних типів

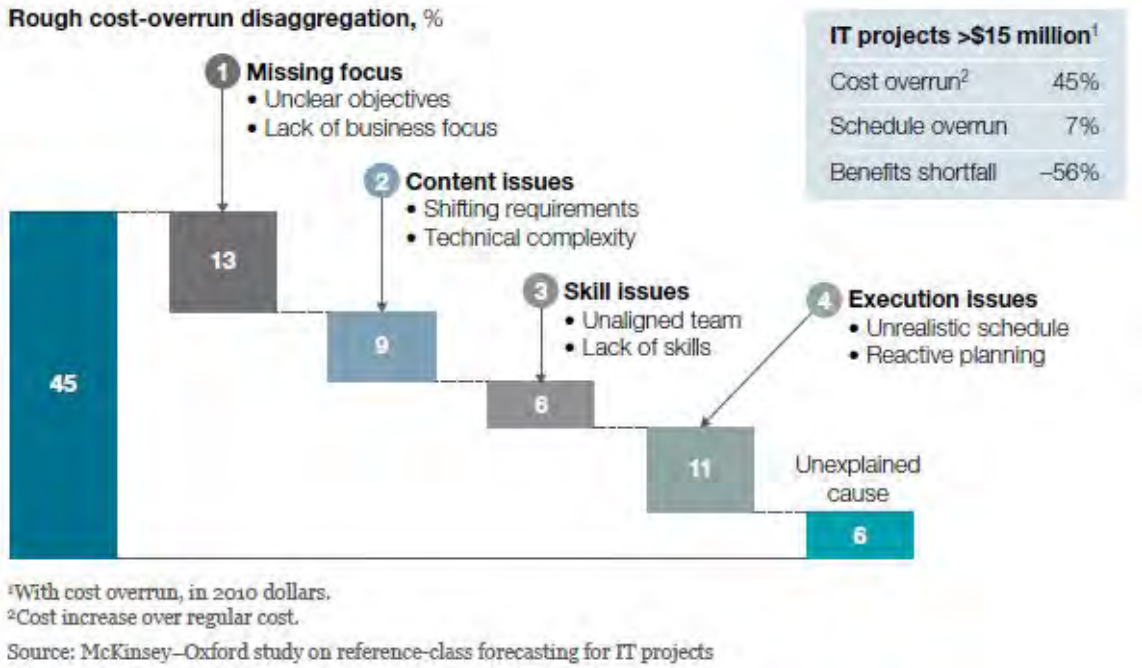


Рис. 3. Чотири групи питань, які викликають більшість невдач програмних проектів

Отже, розроблення ПЗ не завжди завершується успішно, і часто потрапляння програмного проекту в категорію проблемних пов'язане із невиконанням запланованих термінів виконання проекту або перевищення бюджету, тобто може бути обумовлене некоректною оцінкою трудомісткості розроблення ПЗ.

Тому *актуальною задачею* наразі є підвищення достовірності оцінювання трудомісткості розроблення ПЗ.

Значущість оцінки розміру програмного коду
при оцінюванні трудомісткості розроблення програмного забезпечення

Одним з основних методів оцінки трудомісткості програмного проекту є алгоритмічне моделювання [10] – метод, при якому визначається залежність трудомісткості проекту від якогось кількісного показника ПЗ (як правило, це розмір коду). Проводиться оцінка цього показника для даного проекту, після чого за допомогою моделі прогнозуються майбутні витрати. Більшість моделей для визначення трудомісткості розроблення ПЗ можуть бути зведені до функції п'яти основних параметрів [10–12]:

1) розміру кінцевого продукту – як правило, кількість рядків коду або кількість функційних точок, необхідних для реалізації даної функційності;

2) особливостей процесу, використовуваного для одержання кінцевого продукту, зокрема, його здатність уникати непродуктивних видів діяльності (переробок, витрат на взаємодію);

3) можливостей персоналу, який бере участь у розробленні ПЗ, особливо його професійного досвіду та знання предметної галузі проекту;

4) середовища, яке складається з інструментів та методів, використовуваних для ефективного виконання розроблення ПЗ та автоматизації процесу;

5) потрібної якості продукту, яка включає в себе його функційні можливості, продуктивність, надійність та адаптованість.

Найбільш впливовим фактором оцінки трудомісткості в цих моделях є розмір програмного забезпечення [10–12]. Основними одиницями вимірювання розміру ПЗ є: кількість рядків коду (Lines of Code –LOC) та функційні точки (Function Points – FP). Кількість рядків коду (LOC-оцінка) – найвідоміша, найпоширеніша та найбільш вживана одиниця вимірювання [2, 4, 10].

Переваги використання LOC-оцінки в якості одиниць вимірювання [2, 4, 10]: широке поширення та легка адаптованість; можливість співставлення методів вимірювання розмірів та продуктивності в різних групах розробників; безпосередній зв'язок з кінцевим продуктом; легка оцінка до завершення проекту; оцінка розмірів ПЗ на основі точки зору розробника – фізична оцінка створеного продукту.

Поряд з перевагами, *застосування LOC має й ряд проблем* [2, 4, 10]: LOC-оцінка важка для застосування при оцінювання розміру ПЗ на ранніх етапах розроблення; рядки сирцевого коду можуть розрізнятись в залежності від типів мов програмування, методів проектування, стилю та здібностей програміста; показники LOC не можуть застосовуватись при здійсненні нормалізації у випадку, якщо застосовувані платформи розроблення або мови є різними; застосування методів оцінки за допомогою підрахунку кількості рядків коду не регламентується промисловими стандартами; розроблення ПЗ може бути пов'язане із великими витратами, які прямо не залежать від розмірів програмного коду – складання специфікацій вимог та документів користувача, не включених у прямі витрати на кодування; програмісти можуть бути незаслужено премійовані за досягнення високих показників LOC у випадку, якщо керівництво помилково вважатиме це ознакою високої продуктивності, але при цьому відсутнім буде ретельно розроблений проект (сирцевий код не є самоціллю при створенні продукту – головну роль мають функційні властивості та показники продуктивності); генератори коду часто продукують надмірний обсяг коду, в результаті чого спотворюються LOC-оцінки; при підрахунку кількості рядків коду слід розрізняти автоматично та вручну створений код.

Тоді *метою даного дослідження* є розроблення методу оцінювання кількості автоматично генерованого програмного коду для підрахунку реальної кількості рядків коду, створеного саме програмістом, яка дасть можливість більш точно і достовірно оцінити трудомісткість розроблення ПЗ.

Метод оцінювання кількості автоматично генерованого програмного коду

До складу більшості середовищ розроблення входять набори типових елементів. Щоразу коли створюється новий проект, відповідне середовище розроблення автоматично створює «скелет» майбутнього додатку, причому цей код можна негайно безпомилково скомпілювати і запустити на виконання. Програмний проект в такому випадку містить автоматично генерований програмний код, який представляє собою основу майбутньої програми. Більш складні елементи управління мають так звані «програми-майстри» (Wizards), які допомагають налаштувати поведінку елементів управління, автоматично генеруючи код в залежності від обраних опцій. Автоматична генерація програмного коду економить час розробників, позбавляє їх необхідності щоразу заново створювати типовий програмний код і, звісно, призводить до зменшення трудомісткості розроблення ПЗ. Саме з метою правильного визначення LOC-оцінки програмного коду і подальшої достовірної оцінки трудомісткості розроблення ПЗ, і виникає необхідність розрізняти автоматично генерований програмний код і програмний код, написаний вручну програмістом.

Будь-який програмний код може бути представлений у вигляді об'єднання множин автоматично генерованого програмного коду та програмного коду, написаного вручну програмістом:

$$SSC = ASC \cup HSC, \tag{1}$$

де ASC – автоматично генерований програмний код, HSC – код, написаний програмістом вручну.

Тоді LOC-оцінку програмного коду представимо в наступному вигляді:

$$LOC = LOC_{auto} + LOC_{by\_hand}, \tag{2}$$

де LOC_{auto} – кількість рядків автоматично генерованого коду, LOC_{by_hand} – кількість рядків коду,

написаного програмістом вручну.

Рядки автоматично генерованого коду складаються з певних елементів (конструкцій), які утворюють наступну множину (ця множина буде мати різне наповнення для різних мов та середовищ програмування):

$$AC = \{el_1, el_2, ..., el_n\}, \quad (3)$$

де el_i – i-й елемент (конструкція) коду, який генерується автоматично (основним обмеженням на такий елемент є те, що він повинен складати один рядок коду), n – кількість елементів (конструкцій), які можуть генеруватись автоматично (кількість n є різною для різних середовищ та мов програмування).

Весь програмний код представимо у вигляді множини його рядків:

$$SC = \{line_1, line_2, ..., line_{LOC}\}, \quad (4)$$

де $line_j$ – j-й рядок коду, причому $line_j$ – це підмножина, яка складається з конструкцій певної мови програмування, букв, цифр та символів, доупстимих алфавітом тієї чи іншої мови програмування. Основною вимогою до програмного коду є обов'язкове дотримання правил CodeStyle, зокрема, в частині оформлення програмного коду (нова конструкція пишеться з нового рядка і т.і.).

Розробимо універсальні *продукційні правила оцінювання кількості рядків автоматично генерованого програмного коду*:

1) якщо $el_1 \in line_1$, то $LOC_{auto_1} = LOC_{auto_1} + 1$;

2) …

3) якщо $el_i \in line_1$, то $LOC_{auto_i} = LOC_{auto_i} + 1$;

4) …

5) якщо $el_n \in line_1$, то $LOC_{auto_n} = LOC_{auto_n} + 1$;

6) …

7) якщо $el_1 \in line_j$, то $LOC_{auto_1} = LOC_{auto_1} + 1$;

8) …

9) якщо $el_i \in line_j$, то $LOC_{auto_i} = LOC_{auto_i} + 1$;

10) …

11) якщо $el_n \in line_j$, то $LOC_{auto_n} = LOC_{auto_n} + 1$;

12) …

13) якщо $el_1 \in line_{LOC}$, то $LOC_{auto_1} = LOC_{auto_1} + 1$;

14) …

15) якщо $el_i \in line_{LOC}$, то $LOC_{auto_i} = LOC_{auto_i} + 1$;

16) …

17) якщо $el_n \in line_{LOC}$, то $LOC_{auto_n} = LOC_{auto_n} + 1$.

Розроблені правила є універсальними для кожної мови та середовища програмування, але наповнення множини $AC = \{el_1, el_2, ..., el_n\}$ буде різним (індивідуальним) для кожної конкретної мови та середовища програмування.

На основі розроблених продукційних правил підрахунку кількості рядків автоматично генерованого програмного коду розробимо *метод оцінювання кількості автоматично генерованого програмного коду*:

1) відбувається формування множини елементів (конструкцій) $AC_{concrete} = \{el_{1_{concrete}}, el_{2_{concrete}}, ..., el_{n_{concrete}}\}$, які генеруються автоматично, для конкретної мови (середовища) програмування;

2) за методом здійснення пошуку в ширину в прямому напрямку [12] в множині продукційних правил відбувається пошук правила для кожного з елементів множини $AC_{concrete}$, згідно з якими ведеться підрахунок лічильників кількості рядків (використань) кожного автоматично генерованого елементу (конструкції);

3) виконується оцінювання кількості автоматично генерованого програмного коду:

$$LOC_{auto} = \sum_{i=1}^{n} LOC_{auto_i}, \quad (5)$$

де LOC_{auto_i} – кількість рядків (використань) кожного автоматично генерованого елементу (конструкції) конкретної мови програмування; n – кількість елементів (конструкцій), які можуть генеруватись автоматично (кількість n є різною для різних середовищ та мов програмування);

4) виконується оцінювання кількості рядків коду, написаного програмістом вручну:

$$LOC_{by\_hand} = LOC - LOC_{auto}, \quad (6)$$

де LOC – кількість рядків коду (наразі є багато інструментів, які надають LOC-оцінку програмного коду); LOC_{auto} – кількість рядків автоматично генерованого коду, підрахована за формулою (5).

Розроблені правила і метод оцінювання кількості автоматично генерованого програмного коду дають можливість не лише порахувати кількість рядків автоматично генерованого програмного коду, але й дозволяють підрахувати кількість використань кожного елементу (конструкції), яка генерується автоматично, а також кількість рядків коду, написаного вручну програмістом.

Висновки

Розроблення ПЗ не завжди завершується успішно в зв'язку з недотриманням запланованих термінів виконання проекту або перевищення бюджету, що часто обумовлено некоректною оцінкою трудомісткості розроблення ПЗ. Одним з основних методів оцінки трудомісткості розроблення ПЗ є алгоритмічне моделювання – метод, при якому визначається залежність трудомісткості проекту від якогось кількісного показника ПЗ. Найбільш впливовим фактором оцінки трудомісткості в цих моделях є кількість рядків коду або LOC-оцінка.

Як показав проведений аналіз, використання LOC-оцінки має ряд проблем, серед яких спотворення LOC-оцінок через неможливість відокремлення кількості рядків коду, який буде генеруватись автоматично середовищем розроблення, від кількості рядків коду, який буде створюватись вручну програмістом.

В статті розроблено продукційні правила і метод оцінювання кількості автоматично генерованого програмного коду, які забезпечують:

1) оцінювання кількості рядків автоматично генерованого програмного коду;

2) оцінювання кількості використань кожного автоматично генерованого елементу (конструкції) для конкретної мови програмування;

3) оцінювання кількості рядків коду, написаного вручну програмістом.

Перспективою для подальших досліджень авторів є розроблення програмного засобу оцінювання кількості автоматично генерованого програмного коду, який буде базуватись на розроблених у статті продукційних правилах і методі оцінювання кількості автоматично генерованого програмного коду. Даний програмний засіб матиме базу знань, яка міститиме множини автоматично генерованих елементів (конструкцій) декількох мов програмування, записані в неї розробниками програмного засобу, а також множини автоматично генерованих елементів (конструкцій) для інших (невідомих засобу) мов програмування, записані в неї в процесі роботи користувача з програмним засобом, тобто користувач матиме можливість вибрати мову серед запропонованих програмним засобом або ввести автоматично генеровані елементи іншої мови програмування (доповнити таким чином базу знань засобу). Крім цього, база знань засобу міститиме розроблені продукційні правила оцінювання кількості автоматично генерованого програмного коду. Програмний засіб здійснюватиме аналіз програмного коду і, на основі використання розробленого методу, оцінюватиме кількість рядків автоматично генерованого програмного коду, кількість використань кожного автоматично генерованого елементу (конструкції) для конкретної мови програмування, а також кількість рядків коду, написаного вручну програмістом.

Література

1. Кикоть О. П. Метод оцінювання трудомісткості розроблення програмного забезпечення / О. П. Кикоть, С. В. Козак, Д. О. Денисюк // Інтелектуальні технології в системному програмуванні: Всеукраїнська науково-практична конференція молодих учених та студентів, 22–24 квітня 2015 р. : матеріали конференції. – Хмельницький, 2015. – С. 75–82.

2. Фатрелл Р. Т. Управление программными проектами: достижение оптимального качества при минимуме затрат / Р. Т. Фатрелл, Д. Ф. Шафер, Л. И. Шафер ; пер. с англ. – Москва : Изд. дом «Вильямс», 2003. – 1136 с.

3. Брауде Э. Дж. Технология разработки программного обеспечения / Эрик Дж. Брауде. – СПб : Питер, 2004. – 655 с.

4. Макконнелл С. Совершенный код. Мастер-класс / С. Макконнелл. – Москва : Издательство "Русская редакция", 2013. – 896 с.

5. Шамие К. Системная инженерия для «чайников» / К. Шамие. – New York : John Wiley & Sons, 2014. – 76 с.

6. Зыль С. Проектирование, разработка и анализ программного обеспечения систем реального времени / С. Зыль. – Санкт-Петербург : «БХВ-Петербург», 2010. – 336 с.

7. The Standish Group International: CHAOS Manifesto – Think big, act small. Technical report, CHAOS Knowledge Center (2013).URL: http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf (viewed on February 5, 2018). – Title from the screen.

8. Hastie Shane. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch / Shane Hastie, Stéphane Wojewoda. URL: http://www.infoq.com/articles/standish-chaos-2015 (viewed on February 5, 2018). – Title from the screen.

9. Вендров А. М. Проектирование программного обеспечения экономических информационных

систем : учебник / А. М. Вендров. – Москва : Финансы и статистика, 2006. – 544 с.

10. Sommerville I. Software Engineering / I. Sommerville. – London : Pearson, 2015. – 816 p.

11. Лаврищева Е. М. Software Engineering компьютерных систем. Парадигмы, технологии и CASE-средства программирования / Е. М. Лаврищева. – Киев : Наукова думка, 2013. – 283 с.

12. Гаврилова Т. А. Базы знаний интеллектуальных систем / Т. А. Гаврилова, В. Ф. Хорошевский. – Санкт-Петербург : Питер, 2001. – 384 с.

References

1. O. P. Kykot, S. V. Kozak, D. O. Denysyuk. Metod otsinuvannya trudomistkosti rozroblennya programnogo zabezpechennya. Intelektualni tekhnologii v systemnomu programuvanni: Vseukrainska naukovo-praktychna konferenciya molodyh vchenykh ta studentiv, 22-24 kvitnya 2015: materiali konferencii. – Khmelnytskyi, 2015. – S. 75-82.

2. R.T. Futrell, D.F. Shafer, L.I. Shafer. Quality Software Project Management. – Prentice Hall PTR, 2003. – 1680 p.

3. E. K. Braude. Software Design: From Programming to Architecture. – John Wiley and Sons, 2004. – 550 p.

4. S. McConnell. Code complete. – Microsoft Press, 2013. – 896 p.

5. C. Shamieh. Systems Engineering for Dummies. – Wiley Publishing, 2011. – 74 p.

6. S. Zyl. Proektirovanie, razrabotka i analiz programmnogo obespecheniya sistem realnogo vremeni. – Sankt-Peterburg: BHV-Peterburg, 2010. – 336 s.

7. The Standish Group International: CHAOS Manifesto – Think big, act small. Technical report, CHAOS Knowledge Center (2013) [Electronic resource]: http://www.versionone.com/assets/img/files/CHAOSManifesto2013.pdf (viewed on February 5, 2018). – Title from the screen.

8. Hastie Shane. Standish Group 2015 Chaos Report – Q&A with Jennifer Lynch [Electronic resource] / Shane Hastie, Stéphane Wojewoda. – Mode of access: http://www.infoq.com/articles/standish-chaos-2015 (viewed on February 5, 2018). – Title from the screen.

9. A.M. Vendrov. Proektirovanie programmnogo obespecheniya ekonomicheskih informatsionnih sistem: uchebnik. – Moskva: Finansi i statistica, 2006. – 544 s.

10. Sommerville. Software Engineering / I. Sommerville. – London: Pearson, 2015. – 816 p.

11. Ye. M. Lavrishcheva. Software Engineering komputernyh sistem. Paradigmi, tekhnologii i CASE-sredstva programmirovaniya. – Kiev: Naukova dumka, 2013. – 283 s.

12. T. A. Gavrilova, V. F. Khoroshevskiy. Bazy znaniy intellektualnih sistem. – Sankt-Peterburg: Piter, 2001. – 384 s.

Рецензія/Peer review : 12.02.2018 р. Надрукована/Printed :27.03.2018 р.
Рецензент: д.т.н., проф. Р.В. Сорокатий