ISSN 2219-9365

УДК 004.415.53
DOI: 10.31891/2219-9365-2021-68-2-12

ГУРМАН І. В., ЧЕШУН В. М.
Хмельницький національний університет
ДЖУЛІЙ А. В., ЧОРНЕНЬКИЙ В. І.
Університет економіки і підприємництва

ОЦІНОЧНІ ФУНКЦІЇ І МЕТРИКИ ДЛЯ ВИЯВЛЕННЯ ПОМИЛОК ПРИ ТЕСТУВАННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Значні напрацювання діагності і науковців та створення великої кількості методів тестування програмного забезпечення на сьогодні не вирішили проблему повної локалізації дефектів програмного коду і не зменшили актуальність цієї задачі.

Дослідження присвячене аналізу можливості підвищення ефективності методів автоматизованого тестування програмного забезпечення із застосуванням різних варіантів оціночних функцій і метрик, які широко використовуються для оптимізації тестів і оцінки якості результатів тестування. В роботі розглянуто базові класи методів тестування програмного забезпечення, напрямки і технології автоматизації тестування, проведено аналіз зв'язку функцій придатності, коефіцієнтів подібності та метрик із результатами тестування. Розглянуті коефіцієнти подібності Кульчинського, D2, Tarantula, Rogers&Tanimodo, Ochiai, Barinel, а також метрики Expense, Exam-Score, F3 (Jin і Orso), Laghari, T-Score, Mean Average Precision, Ulysis, G тощо. Від якості обраних функцій і їх відповідності методу тестування значною мірою залежить і результат локалізації дефектів програмного коду, що є передумовою зменшення ризику збоїв в роботі програмного забезпечення, фінансових та моральних збитків розробників та користувачів програмних продуктів.

Хоча більшість функцій та метрик орієнтовано на статистичні методи автоматизованого тестування програмного забезпечення на основі спектру, вони можуть бути використані або адаптовані до застосування і в інших методах.

Ключові слова: тестування програмного забезпечення, функція придатності, коефіцієнт подібності, метрика.

I. GURMAN, V. CHESHUN
Khmelnytskyi National University
A. DZHULIY, V. CHORNENKYI
University of Economics and Entrepreneurship

EVALUATION FUNCTIONS AND METRICS FOR ERROR DETECTION WHEN TESTING SOFTWARE

Significant achievements of diagnostics and scientists and the creation of a large number of software testing methods to date have not solved the problem of complete localization of software code defects and have not reduced the relevance of this task.

The study is devoted to the analysis of the possibility of improving the efficiency of automated software testing methods using different options of evaluation functions and metrics, which are widely used to optimize tests and assess the quality of test results. The paper considers the basic classes of software testing methods, directions and technologies of test automation, analyzes the relationship of suitability functions, similarity coefficients and metrics with test results. The similarity coefficients of Kulchinsky, D2, Tarantula, Rogers & Tanimodo, Ochiai, Barinel, as well as metrics Expense, Exam-Score, F3 (Jin and Orso), Laghari, T-Score, Mean Average Precision, Ulysis, G, etc. are considered. Features of the selected functions and their compliance with the testing method largely determines the result of localization of software code defects, which is a prerequisite for reducing the risk of software failures, financial and moral losses of software developers and users.

Although most functions and metrics focus on statistical methods of automated spectrum-based software testing, they can be used or adapted for use in other methods.

Key words: software testing, suitability function, similarity coefficient, metrics.

Вступ. Постановка проблеми. Стрімка інформатизація суспільства відбувається у безпосередньому зв'язку з розвитком мов і технологій створення програмних продуктів. ІТ-галузь сьогодні виходить на лідерські позиції в економіках багатьох країн і потребує все більше фахівців, а жорстка конкуренція призводить до інтенсифікації їх роботи. Незважаючи на постійне вдосконалення інструментів і технологій створення програмних продуктів, відмовитись від участі людини в його розробці на сьогодні неможливо, що зумовлює вплив людського фактору на отримуваний результат і неминуче виникнення помилок в програмних продуктах.

Дослідження, проведене ще в 2002 році, показало, що лише для економіки США дефекти програмного забезпечення становили щорічні витрати в 60 мільярдів доларів, і з тих пір спостерігається тенденція до зростання ціни помилок програмування [1]. Подекуди дефекти програмного забезпечення мають катастрофічні наслідки, чому прикладом може слугувати вибух в червні 1982 року силою в три кілотонни на газопроводі в Сибіру, втрата Європейським космічним агентством в червні 1996 року ракети-носія Ariane 5, яка була розроблена «Чорний понеділок» фондової біржі Wall Street 19 жовтня 1987 року тощо [2].

Проблеми наявності помилок програмного коду, поряд із високими вимогами щодо функціональності і надійності програмних продуктів, загострюють питання забезпечення їх якості. Одним із основних

інструментів для цього стає тестування, що актуалізує завдання розробки нових ефективних методів тестування програмних продуктів та вдосконалення існуючих.

Аналіз останніх досліджень і публікацій. Тестування програмного забезпечення проходить протягом усього циклу розробки програмного забезпечення і, традиційно, потребує понад 50% зусиль, що витрачаються на розробку та обслуговування програмного продукту [4, 5]. Локалізація дефектів у програмному забезпеченні спрямована на виявлення помилок та несправних функцій у програмному продукті та зменшення ризиків збоїв у його роботі і є однім із найбільш складних, трудомістких, виснажливих та витратних заходів у налагодженні програм [4, 6]. Це зумовило появу великої кількості методів локалізації, також ідентифікованих як методи тестуванні програмного забезпечення.

Базовими вважаються такі методи локалізації дефектів, як журналювання програми або введення операторів друку, застосування тверджень, введення точок зупину та профілювання [1, 3], до яких в [5] додається метод покриття коду. Ці методи, зазвичай, класифікуються «ручними» і спираються на досвід розробника або експерта. Ефективність «ручного» керування тестуванням програмного забезпечення в сучасних умовах визнана малоефективною. Це пов'язано з тим, що виявлення, локалізації та усунення дефектів програмного коду є процесом нетривіальним і схильним до помилок прийняття рішень, в якому навіть досвідчені розробники помиляються майже в 90% випадків у своїх початкових припущеннях, намагаючись визначити причину неочікуваного спрацьовування програми, яке відхиляється від планованого [1].

Автоматизація тестування програмного забезпечення відбувається за різними напрямками. В [7] пропонується перелік основних напрямків автоматизації: автоматизація управління тестуванням; автоматизоване тестування; автоматизація кросбраузерного тестування; автоматизація навантажувального тестування; автоматизація відслідковування помилок; автоматизація тестування API. Серед основних технологій автоматизованого тестування в [8] вказуються приватні рішення; тестування під керуванням даними; тестування під керуванням ключовими словами; використання фре ймворків; запис та відтворення; поведінкове тестування. Скласти повний перелік методів автоматизованого тестування програмного забезпечення, а, тим більш, засобів реалізації цих методів неможливо. В [9] наведено базову класифікацію методів Вонга, яка ділить їх на методи локалізації дефектів на основі спектру, на основі фрагментування програм, на основі стану та на основі машинного навчання.

Хоча наявні різні напрямки автоматизації тестування програмного забезпечення та існує значне різноманіття самих методів тестування, завдання ефективної локалізації помилок не є вирішеним. Свідченням цього є помилки в кодах і збої в роботі програмного забезпечення провідних розробників програмного забезпечення, які при розробці програмного забезпечення велику кількість ресурсів витрачають на налагодження. Всім знайомі повідомлення Microsoft про збої в роботі програмного забезпечення з пропозицією відправити звіт розробнику. За статистикою syzbot за період з листопаду 2017 року по квітень 2021 року було виявлено 979 помилок ядра Linux. Суттєва кількість помилок і збоїв в роботі програмного забезпечення фіксується незважаючи на те, що діяльність з тестування та налагодження може легко коливатися від 50 до 75 відсотків загальної вартості розробки [1].

Здебільшого тестувальники-практики використовують типові функції і метрики, передбачені для реалізації певного методу пошуку і локалізації дефектів. В наукових робот спостерігається тенденція до систематизації і аналізу властивостей оціночних функцій та метрик, а також до дослідження ефективності методів тестування при застосуванні різних критеріїв та способів оцінювання ефективності елементів та результатів діагностичних експериментів [5, 9, 12]. Цьому питанню приділено значно менше уваги і наукових робіт, ніж створенню самих методів тестування.

Метою роботи є визначення та аналізі типових оціночних функцій і метрик, за рахунок яких при тестуванні програмного забезпечення виконуються оцінка якості і оптимізація тестів, а також оцінка якості результатів тестування.

Виклад основного матеріалу. Застосування ефективних методів оцінювання параметрів і результатів діагностичних експериментів є основою для ефективного управління тестовими випробуваннями та їх оптимізації, а також є передумовою для максимального покриття наявних дефектів і їх усунення.

На сьогодні найбільшого поширення набули методи автоматизованого тестування на основі спектру, спрямовані на ефективну локалізацію несправних компонентів за допомогою дослідження поведінки програми [6]. Це робиться шляхом збирання шаблонів виконання різних комбінацій компонентів і відповідних результатів у спектр. Методи локалізації на основі спектру виявилися надзвичайно корисними для локалізації несправностей у великих кодових базах [11]. Спектральні методи спрямовані на виявлення дефектів (хибних командних рядків, функцій або фрагментів програми) на основі статистичного аналізу тестових спектрів. Спектри містять інформацію патерну активності кожного тесту та дають змогу виділити провальні тести як результат. При локалізації помилок коду на основі спектру відбувається ідентифікація оператора (твердження) із шаблоном виконання, максимально наближеним до шаблону збоїв усіх тестових випадків [9].

Ефективна локалізація дефектів програмного забезпечення в таких методах сильно залежить від якості спектрів [6, 11, 16], яка, в свою чергу залежить від математичного підґрунтя процедур синтезу тестів.

Набори тестів можуть створюватися вручну або за допомогою методів автоматичного генерування тестів [Campos et al., 2014]. Генератор набору тестів може створити набір тестів *T*, оптимізуючи функцію придатності *f*, яка приймається як міра якості набору тестів [6]. Оператор визнається більш підозрілим, якщо шаблон його виконання подібний до шаблонів збою всіх невдалих тестових випадків. Оператор визнається менш підозрілим, якщо шаблон його виконання оператора подібний до шаблону виконання успішних тестових випадків. Рівень підозрілості оператора характеризується коефіцієнтом подібності, який обчислюється за наявними статистичними даними результатів тестування. На сьогодні розроблена велика кількість оціночних коефіцієнтів подібності, жоден з яких не претендує на повноту покриття і універсальність. Зокрема, в [5, 12] приводиться математичний опис більше 30 таких коефіцієнтів, що далеко не є повним переліком.

Розглянемо деякі типові приклади оціночних коефіцієнтів подібності.

Для представлення розрахункових формул обчислення коефіцієнтів подібності скористаємося описами типових розрахункових значень, запропонованими в [12]: N – загальна кількість успішно пройдених тестів; N_S – загальна кількість успішно пройдених тестів; N_F – загальна кількість невдалих тестів; N_C – загальна кількість тестів, що охоплюють підозріле твердження; N_U – загальна кількість тестів, що не охоплюють підозріле твердження; N_{CS} – кількість успішно пройдених тестів, що охоплюють підозріле твердження; N_{US} – кількість успішно пройдених тестів, що не охоплюють підозріле твердження; N_{CF} – кількість невдалих тестів, що охоплюють підозріле твердження; N_{UF} – кількість невдалих тестів, що не охоплюють підозріле твердження.

Стосовно наведених значень можна вказати їх взаємозв'язок:

$$N_S = N_{CS} + N_{US}; \tag{1}$$

$$N_F = N_{CF} + N_{UF}; \tag{2}$$

$$N_U = N_{US} + N_{UF}; \tag{3}$$

$$N_C = N_{CS} + N_{CF}; \tag{4}$$

$$N = N_S + N_F = N_C + N_U. \tag{5}$$

З урахуванням формул 1-5 значення загальних кількостей N, N_S, N_F, N_C і N_U можуть бути замінені в розрахункових формулах коефіцієнтів на суми їх складових.

Коефіцієнт Кульчинського [5, 12]:

$$K_K = \frac{N_{CF}}{N_{CS} + N_{UF}}. \tag{7}$$

Коефіцієнт D^2 [5]:

$$K_{D^2} = \frac{N_{CF}^2}{N_{CS} + N_{UF}}. \tag{8}$$

Коефіцієнт Tarantula [11]:

$$K_T = \frac{\frac{N_{CF}}{N_{CF} + N_{UF}}}{\frac{N_{CF}}{N_{CF} + N_{UF}} + \frac{N_{CS}}{N_{CS} + N_{US}}} = \frac{\frac{N_{CF}}{N_F}}{\frac{N_{CF}}{N_F} + \frac{N_{CS}}{N_S}}. \tag{6}$$

Коефіцієнт Rogers&Tanimodo [12]:

$$K_{R\&T} = \frac{N_{CF} + N_{US}}{N_{CF} + N_{US} + N_{CS} + N_{UF}} = \frac{N_{CF} + N_{US}}{N}. \tag{9}$$

Коефіцієнт Ochiai [11]:

$$K_O = \frac{N_{CF}}{\sqrt{(N_{CF} + N_{UF}) \times (N_{CF} + N_{CS})}} = \frac{N_{CF}}{\sqrt{N_F \times (N_{CF} + N_{CS})}}. \quad (10)$$

Коефіцієнт Barinel [11]:

$$K_B = 1 - \frac{N_{CS}}{N_{CS} + N_{CF}} = 1 - \frac{N_{CS}}{N_C}. \quad (11)$$

Незважаючи на різноманіття функцій коефіцієнтів, з результатів досліджень найбільш розповсюджених коефіцієнтів [11] слідує, що жоден із коефіцієнтів подібності не забезпечує статистично значущої різниці порівняно з іншими при використанні для локалізації несправностей на основі спектру, тому їх вибір є прерогативою тестувальника або розробника засобів реалізації методу тестування тощо.

Оцінки підозрілості тверджень в програмному коді широко застосовуються для оцінки ефективності тестування. Існують різні метрики оцінки якісних характеристик тестового процесу.

Метрика Expense [13] реалізується за ранжуванням тверджень відповідно до їх рівня підозрілості використовується для оцінки відсотку програми, що не потребує перевірки для виявлення помилки:

$$M_{\text{Expense}} = \frac{K - k}{K}, \quad (12)$$

де k - це ранг твердження з помилкою у звіті про локалізацію несправності, а K - загальна кількість тверджень у програмі.

Метрика Exam-Score [5, 11, 13] є однією із найпопулярніших і обчислює відсоток коду, який необхідно перевірити. Метрика Exam-Score базується на використанні тих самих параметрів ранжування тверджень, що і метрика Expense:

$$M_{\text{Exam-Score}} = \frac{k}{K}. \quad (13)$$

Окремим класом метрик оцінки якості тестових випробувань є метрики затрати зусиль на тестування, прикладами яких є описані в [14] метрика F3 (Jin і Orso) та її модифікація від Laghari.

У метриці F3 витрачені зусилля оцінюють, вказуючи кількість вірних тверджень, які в середньому мають бути перевірені, перш ніж буде виявлено помилкове твердження:

$$M_{\text{F3}} = m + l + 1, \quad (14)$$

де m – кількість тверджень без помилки, яким присвоєно вищий бал підозрілості, ніж твердженню з помилкою; l – – кількість тверджень без помилки, яким присвоєно такий самий бал підозрілості, як і твердженню з помилкою. У модифікованій метриці F3 від Laghari при оцінці ранжується вага помилок на підставі урахування їх «грубості»:

$$M_{\text{Laghari}} = m + \frac{l+1}{2}. \quad (15)$$

Метрика T-Score [5, 11] орієнтована на використання нестатистичних методів локалізації помилок, які передбачають невеликий набір підозрілих тверджень у програмі. Метрика оцінює відсоток коду, який розробник не повинен перевіряти, перш ніж виявляти помилкове твердження. Метрика використовує граф-модель (PDG) залежності програми для обчислення набору вершин у графі, які необхідно перевірити, щоб досягти помилкового твердження. Найменший набір вершин, що містить помилкове твердження, називається сферою залежності (DS):

$$M_{\text{T-Score}} = \frac{|DS|}{|PDG|}, \quad (16)$$

де $|PDG|$ - загальна кількість вершин в граф-моделі діагностичного процесу; $|DS|$ - кількість вершин в граф-моделі, що входять в сферою залежності.

Метрика усередненого значення точності (Mean Average Precision – MAP) [14] запозичена для пошуку помилок в програмному коді із методів пошуку інформації. MAP визначається шляхом обчислення середнього значення точності:

$$M_{\text{MAP}} = \frac{1}{M}\sum_{i=1}^{K} P(i)\,rel(i), \tag{17}$$

де M - кількість тверджень з помилкою у звіті про локалізацію несправності; K - загальна кількість тверджень у програмі; $P(i)$ – оцінка помилки i-го твердження (точність i-го твердження); $rel(i)$ – бінарний індикатор, що вказує, містить i-те твердження помилку чи ні.

Подібна метрика обґрунтовується в [15] при аналізі підходу до тестування на критеріях щільності-різноманіття-унікальності (DDU), за якими створюються «хороші» набори тестів шляхом покращення певних структурних властивостей спектрів. Метрика описується під назвою Ulysis:

$$M_{\text{Ulysis}} = \sum_{i=1}^{K} P(i)W(i), \tag{18}$$

де $W(i)$ – бінарний індикатор, який оцінює підозрілість i-го твердження на помилку ($W(i)$=0 – якщо найвищий ранг підозрілості, характерний для i-го твердження; $W(i)$=1 – якщо найвищий ранг підозрілості, нехарактерний для i-го твердження).

В [10] для реалізації тестування за методом DDU пропонується використати альтернативну метрику, що базується на використанні індексу Джіні-Сімпсона для вимірювання різноманітності (G). Метрика G обчислює ймовірність того, що два випадково вибраних елементи тесту будуть різних видів:

$$M_{\text{G}} = 1 - \frac{\sum_{i=1}^{K} k\times(k-1)}{K\times(K-1)}, \tag{19}$$

де k – кількість тестів, які мають однакову активність (здатність виявляти дефекти).

Висновки

Розглянуті оціночні функції і метрики є типовими прикладами математичного апарату, застосовуваного для оптимізації процесу тестування програмного забезпечення, підвищення його ефективності та оцінювання якості отриманих результатів. Від якості обраних функцій і їх відповідності методу тестування значною мірою залежить і результат локалізації дефектів програмного коду, що є передумовою зменшення ризику збоїв в роботі програмного забезпечення, фінансових та моральних збитків розробників та користувачів програмних продуктів. Хоча більшість розглянутих функцій та метрик орієнтовано на статистичні методи автоматизованого тестування програмного забезпечення на основі спектру, але можуть бути використані або адаптовані до застосування і в інших методах.

References

1. Alexandre Perez, Rui Abreu, Eric Wong A survey on fault localization techniques. Publication date: 2014. [Elektronnyi resurs]. – Rezhym dostupu: https://inlnk.ru/megw7
2. Tsina pomylky: naidorozhchi kompiuterni bahy v istorii IT [Elektronnyi resurs]. – Rezhym dostupu: https://www.vectornews.net/news/society/66325-cna-pomilki-naydorozhch-kompyutern-bagi-v-storyi-t.html
3. W. Eric Wong, Vidroha Debroy. A Survey of Software Fault Localization: Technical Report UTDCS-45-09. Department of Computer Science The University of Texas at Dallas, November 2009. 19 p.
4. Rongxin Wu, Hongyu Zhang, Shing-Chi Cheung, Sunghun Kim. CrashLocator: locating crashing faults based on crash stacks. ISSTA 2014: Proceedings of the 2014 International Symposium on Software Testing and AnalysisJuly 2014. P. 204–214. DOI: https://doi.org/10.1145/2610384.2610386
5. Xiao Hong Su, Dan Dan Gong, Tian Tian Wang, Pei Jun Ma. A Survey of Automated Software Fault Localization Approach. Applied Mechanics and Materials, May 2014, Volumes 556-562. P. 6102-6105.
6. Prantik Chatterjee, Abhijit Chatterjee, Jose Campos, Rui Abreu. Diagnosing Software Faults Using Multiverse Analysis. Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20), 2020. P. 1629-1635.
7. Yehorova O.V. Prohramni zasoby dlia testuvannia prohramnoho zabezpechennia / O.V. Yehorova, V.P. Bychok // Molodyi vchenyi, Tekhnichni nauky. – 2019. – № 11 (75). – S. 680-684.
8. Analiz instrumentiv dlia avtomatyzovanoho testuvannia prohramnoho zabezpechennia / Olha Melkozerova, Aleksei Narezhnyi, Serhei Malakhov. // Kompiuterni nauky ta kiberbezpeka. – 2019. – № 1. – S.75-84. DOI: https://doi.org/10.26565/2519-2310-2019-1-07
9. W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. A survey on software fault localization. IEEE Transactions on Software Engineering, 42(8), 2016. P.707–740.
10. Alexandre Perez, Rui Abreu, Arie van Deursen A test-suite diagnosability metric for spectrum-based fault localization approaches. 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). P. 654-664.

ISSN 2219-9365

11. Spencer Pearson, Jose´ Campos, Rene´ Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), I EEE, 2017. P. 609–620. DOI: https://doi.org/10.1109/ICSE.2017.62

12. W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. The dstar method for effective software fault localization. IEEE Transactions on Reliability, 63(1):, IEEE, 2013. P. 290–308.

13. James A Jones and Mary Jean Harrold. Empirical evaluation of the tarantula automatic fault-localization technique. In Proceedings of the 20th IEEE/ACM international Con- ference on Automated software engineering, 2005. P. 273–282

14. Aaron Ang, Alexandre Perez, Arie van Deursen, Rui Abreu. Revisiting the practical use of automated software fault localization techniques. In 2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). IEEE, 2017. P. 175–182.

15. Diego Calvanese, Julien Corman, Davide Lanti, and Simon Razniewski Counting Query Answers over a DL-Lite Knowledge Base International Joint Conference on Artificial Intelligence, 2020. P. 1629–1635.