УДК 004.415.53

Малишевський О.Г.

ПРІОРИТЕЗАЦІЯ ТЕСТІВ ЯК МЕТОД ШВИДКОГО ВИЯВЛЕННЯ СЕРЙОЗНИХ ПОМИЛОК

У роботі була розглянута пріоритезація, яка враховує вартість тестів і серйозність помилок. На відміну від попередніх досліджень, серйозність помилок була заснована на їхньому впливі на надійність програми. Декілька програм було використано як об'єкти дослідження. В одній з них були використані як справжні дані про вартість тестів, так і справжній операційний профіль. Вартість тестів і серйозність помилок були досліджені в чотирьох різних шкалах і вивчений їхній вплив на ефективність пріоритезації. Було розглянуто три підходи до впровадження інформації про вартість і критичність тестів у процес пріоритезації та досліджена їх ефективність.

Ключові слова: *тест, тестування програмного забезпечення, метрики APFD та $APFD_c$, шкала, регресивне тестування.*

В работе была рассмотрена приоритезация, которая учитывает стоимость тестов и серьезность ошибок. В отличие от предыдущих исследований, серьезность ошибок была основана на их влиянии на надежность программы. Несколько программ было использовано в качестве объектов исследования. В одной из них были использованы как настоящие данные о стоимости тестов, так и настоящий операционный профиль. Стоимость тестов и серьезность ошибок были исследованы в четырех разных шкалах и изучено их влияние на эффективность приоритезации. Были рассмотрены три подхода ко внедрению информации о стоимости и критичности тестов в процесс приоритезации и исследована их эффективность.

Ключевые слова: *тест, тестирование програмного обеспечения, метрики APFD и $APFD_C$, шкала, регрессивное тестирование.*

It was in-process considered prioritezaciya which takes into account the cost of tests and seriousness of errors. Unlike previous researches, the seriousness of errors was based on their influence on reliability of the program. A few programs were utillized as research objects. In one of them, were utillized, both real information about the cost of tests and real operating type. The cost of tests, and seriousness of errors, were investigational in four different scales and their influence is studied on efficiency of prioritezaciya. Three going were considered near introduction of information about a cost and criticism of tests in the process of prioritezaciya and their efficiency is investigational.

Key words: *test, testing of software, birth-certificate of apfd and apfdc, scale, regressive testing.*

ВСТУП

Одним з методів перевірки, чи задовольняє програма заданим вимогам та її специфікації є тестування (виконання програми та перевірка її поведінки на відповідність специфікаціям). Кожний тест з набору тестів *T*, який використовується у тестуванні, складається з множини вхідних значень (сценарієв тестування). Звичайно набір тестів створюється, виходячи з деякої множини правил, який має назву критерій адекватності. Цій критерій виражає вимоги, яким повинен задовольняти набір тестів [1]. На стадії супроводження програми багаторазово використовується регресивне тестування, яке перевіряє, що внесені у програму модифікації змінили програму відповідно з новими специфікаціями, виправили знайденні помилки та не внесли нових помилок [1].

У регресивному тестуванні зростає набір тестів, збільшуючи вартість та тривалість тестування. Наприклад, одна з систем ПЗ (20 000 строк коду) потребує сім тижнів для

тестування при ви користуванні усіх тестів у наборі. У багатьох випадках у процесі регресивного тестування можна використовувати тільки підмножину набору тестів для перевірки модифікованої програми. Але іноді буває складно або не допускається використовування неповного набору тестів, наприклад, для програм, надійність яких є критичною (авіоніка або керування медичним обладнанням). У даному випадку для зменшення вартості регресивного тестування може бути використано інший підхід: тести упорядковують (пріоритезують) для регресивного тестування таким чином, щоб більш важливі з них виконувалися у першу чергу.

Питанню пріоритезації приділено багато уваги [2-22]. У методах пріоритезації тести сортують таким чином, щоб більш ефективно досягти заданої мети, наприклад, найбільш швидкого покриття операторів програмного коду, функцій програми у порядку частоти їх використання або підсистем у порядку частоти їх збоїв у минулому. Можлива мета пріоритезації – збільшення швидкості виявлення помилок набором тестів у процесі тестування. Підвищена швидкість виявлення помилок може забезпечити більш ранній зворотний зв'язок з системою що тестується та дозволити розроблювачам почати пошук місцезнаходження помилок, а також їх виправлення раніше, ніж це було б можливо у іншому випадку. Такий зворотній зв'язок забезпечує виявлення ранніх ознак того, що задана мета ще не досягнута, та дозволяє приймати стратегічні рішення про строки реалізації на ранніх етапах. Підвищена швидкість виявлення помилок збільшує ймовірність того, що у випадку передчасного припинення процесу тестування тести, які забезпечують найбільшу здатність виявляти помилки у строки, виділені на тестування, вже були виконані.

У роботах [4, 5, 7, 10, 13, 20] була представлена метрика APFD для виміру швидкості виявлення помилок під час виконання набору тестів у заданому порядку. У цих роботах були представлені декілька методів пріоритезації метою яких являлось збільшення швидкості виявлення помилок, а також була емпірично оцінена їх ефективність. Результати показали, що деякі методи пріоритезації можуть прискорити виявлення помилок.

На практиці як вартість тестів, так і серйозність помилок можуть значно відрізнятися між собою. Тому у наступних роботах [6, 9, 21, 22] була розглянута пріоритезація, враховуюча вартість тестів та серйозність помилок. Також була розроблена метрика $APFD_C$, враховуюча їх, та було проілюстровано застосування пріоритезації.

У даній роботі ми більш детально розглянемо застосування пріоритезації враховуючої вартість тестів та серйозність помилок, розглянемо альтернативний підхід впровадження інформації про серйозність потенційних помилок у процес пріоритезації, розглянемо більш велику кількість методів пріоритезації, використаємо реальну, замість штучно сгенерованої, інформацію про вартості тестів та серйозності помилок, розглянемо їх різноманітні шкали, а також проведемо розширені дослідження у яких використаємо більшу кількість програм-об'єктів дослідження.

МЕТРИКИ APFD ТА $APFD_C$

Для аналізу ефективності пріоритезації необхідно кількісно її оцінити.

У ранніх дослідженнях [4, 5, 7, 10, 13, 20] використовувалась метрика APFD (weighted average of the percentage of faults detected), яка оцінювала швидкість виявлення помилок набором тестів у інтервалі від 0 до 100. Чим більше значення метрики, тим швидше виявляються помилки. Метрика APFD проілюстрована на *Рис. 1. Рис. 1,а* містить таблицю яка показує які помилки (від 1 до 10) виявляє кожний тест (A, B, C, D або E). Чотири частини малюнка (б, в, г та д) графічно зображують метрику для чотирьох різних порядків тестів.

Однак ця метрика базується на двох допущеннях: 1) усі помилки мають ідентичну серйозність та 2) усі тести мають ідентичну вартість. Ці припущення виражаються у тому, що дана метрика просто визначає процент виявлених помилок для виконаної частини набору тестів. Тому у пізніших роботах [6, 9, 21, 22], де враховувалася як вартість тестів так і серйозність помилок, була розроблена та використана метрика $APFD_C$. Ця метрика використовується і у даній роботі.

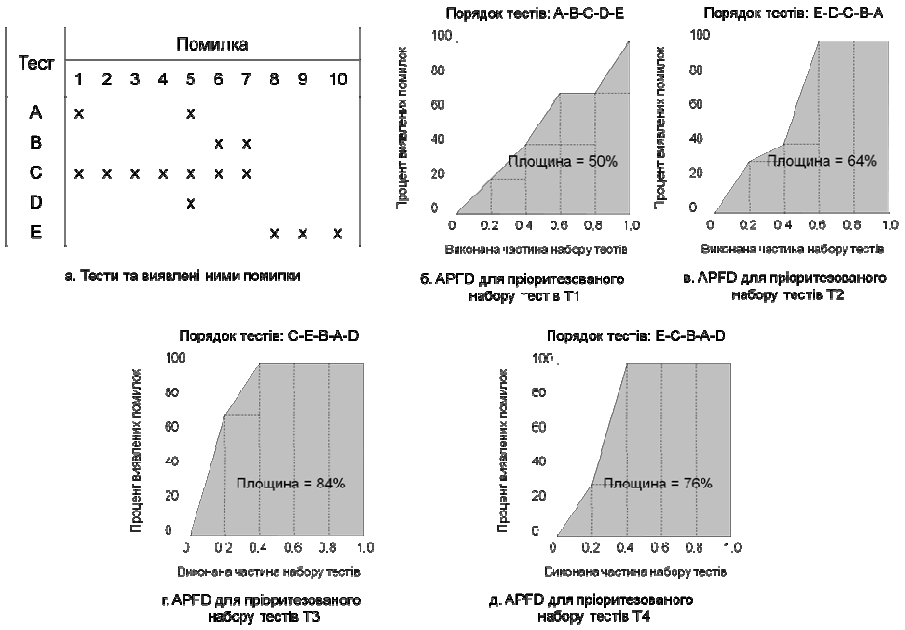

Рис. 1. Приклади, які ілюструють метрику APFD:
а – тести та виявлені помилок; б – APFD для пріори тезованого набору тестів T1;
в – для T2; г – для T3; д – для T4

У метриці $APFD_C$, на графіку, на горизонтальній осі відображається «Процент Сумарної Витраченої Вартості Тестів». Тут, кожен тест у наборі представлений інтервалом вздовж горизонтальної осі, чия довжина пропорційна долі вартості кожного тесту у сумарній вартості атестів у наборі. На вертикальній осі графіка, відображається «Процент Сумарної Серйозності Виявлених Помилок». Тепер, кожна помилка що виявлена набором тестів представлена інтервалом вздовж вертикальної осі, чия висота пропорційна долі її серйозності у загальній сумі серйозності помилок. З точки зору цієї нової інтерпретації, на графіках, внесок тесту зважується у горизонтальному напрямку пропорційно його вартості, а вздовж вертикального напрямку – по сумарній серйозності виявлених їм помилок. Тут, крива обмежує більшу площу для порядку тестів котрий демонструє більше «одиниць-серйозності-помилок-виявлених-на-одиницю-вартості-тесту»; ця площа і становить метрику $APFD_C$.

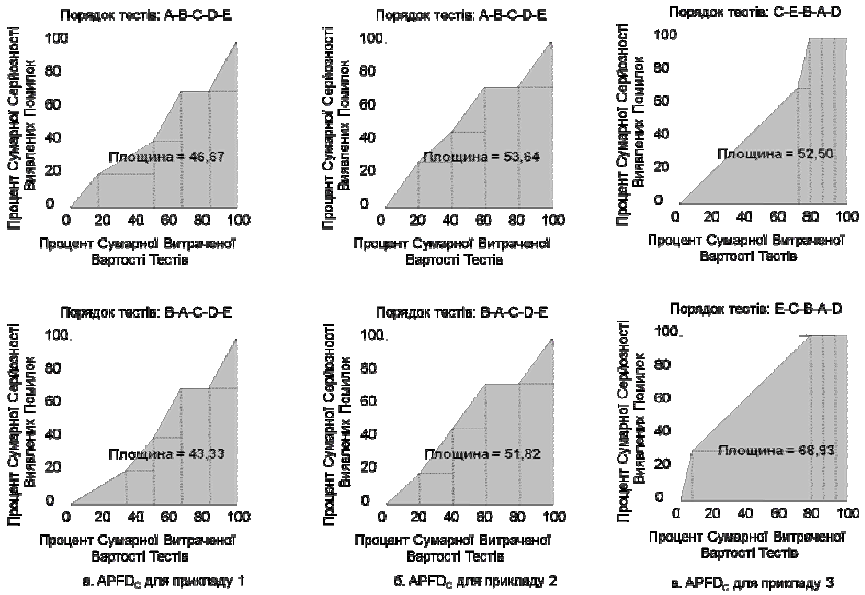

Рис. 2. Приклади що ілюструють метрику APFD

Для ілюстрації метрики $APFD_C$ графічно, *Рис. 2* показує графіки для трьох прикладів. Інформація про виявлення тестами помилок узята з *Рис. 1,а.*

Приклад 1. Розглянемо сценарій, зображений на *Рис. 1*. Припустимо, що вартість тесту B у два рази перевищує вартість тесту A, що потребує дві години машинного часу, тоді як тест A – одну годину. З точки зору помилок, виявлених за годину, порядок тестів A–B–C–D–E має перевагу над порядком B–A–C–D–E, котрий виявляє помилки швидше. Пара графіків яка розташована ліворуч (*Рис. 2*,а) відповідає прикладу 1: верхній графік представляє метрику $APFD_C$ для порядку тестів A–B–C–D–E та нижній графік – метрику $APFD_C$ для порядку B–A–C–D–E. Метрика $APFD_C$ дає перевагу порядку A–B–C–D–E котрий швидше виявляє помилки.

Приклад 2. Працюючи зі сценарієм, показаним на *Рис. 1*, припустимо що усі п'ять тестів мають рівну вартість та що помилки 2…10 мають значення серйозності, що дорівнює *k*, тоді як помилка 1 має серйозність *2k*. У даному випадку тест A виявляє одну більш серйозну помилку та одну менш серйозну, тоді як тест B – тільки дві менш серйозні помилки. З точки зору швидкості виявлення сумарної серйозності помилок порядок тестів A–B–C–D–E має перевагу над порядком B–A–C–D–E. Пара графіків на *Рис. 2,*б що відповідає прикладу 2 показує як метрика $APFD_C$ дає більш високу оцінку порядку тестів котрий раніше виявляє більш серйозну помилку (A–B–C–D–E), при припущенні що помилки 2…10 мають серйозність *k* та помилка 1 має значення серйозності *2k*.

Приклад 3. Припустимо, що усі десять помилок рівнозначні по серйозності та що кожен з тестів A, B, D та E потребує одну годину для виконання, але тест C – десять годин. Однак з точки зору помилок, виявлених за одиницю часу, другий порядок (E–C–B–A–D) має перевагу: он виявляє три помилки протягом першої години та залишається найкращим, ніж перший порядок, до кінця виконання другого тесту. Аналогічний приклад може бути наведений для використання помилки з різною серйозністю при однаковій вартості тестів. Пара графіків на *Рис. 2,в*, що відповідають прикладу 3, показують як метрика відрізняє порядки тестів де тест C має високу вартість: метрика дає більше значення для порядку E–C–B–A–D, ніж для порядку C–E–B–A–D.

Нехай *T* буде набором який складається з *n* тестів з вартостями $t_1, t_2,..., t_n$; *F* – множиною *m* помилок виявлених набором тестів *T*; та $f_1, f_2,..., f_m$ – значеннями серйозності цих помилок. Нехай TF_i буде першим тестом у порядку *T'* набору *T* котрий виявляє помилку *i*. Тоді формула для метрики $APFD_C$ має наступний вигляд:

$$APFD_C = \frac{\sum_{i=1}^{m}\left(f_i \times \left(\sum_{j=TF_i}^{n} t_j - \frac{1}{2} t_{TF_i}\right)\right)}{\sum_{i=1}^{n} t_i \times \sum_{i=1}^{m} f_i}.$$

ВАРТІСТЬ ТЕСТІВ

Існує два питання пов'язаних з вартістю тестів: (1) вимір або оцінка вартості з метою обчислення значення метрики $APFD_C$ для порядку тестів та (2) оцінка вартості для використання у пріоритезації тестів. Вартість тесту зв'язана з ресурсами витраченими на виконання даного тесту та перевірку результатів. Вартість тесту може вимірювати фактичний час витрачений на тестування програми заданим тестом. У іншому випадку, вартість може включати грошові витрати на виконання тесту та перевірку результатів, включаючи вартість обладнання, зарплату, вартість матеріалів необхідних для тестування, втрати доходу зв'язаного з затримкою релізу ПЗ, втрати зв'язані зі зривом строків релізу, та інше.

Визначити вартість тестів після завершення тестування нескладно, що й необхідно зробити для обчислення метрики $APFD_C$. У цьому випадку нам необхідно звернути увагу які ресурси були витрачені на кожний тест. Набагато складніше оцінити вартість тестів перед початком тестування, що необхідно для використання вартості в процесі пріоритезации. У цьому випадку, нам необхідно оцінити вартість тестів перед їх виконанням. Одним способом є аналіз тесту та програмного коду виконаного тестом. Іншим способом, яким ми скористаємося у даній роботі, є використання зібраних даних про вартість тестів на попередніх сесіях тестування (що є можливим при регресивнім тестуванні). Тут ми вважаємо, що вартість тестів не змінюється значно від однієї версії програми до іншої.

У наших експериментах, для регресивного тестування був потрібен лише час виконання кожного тесту (включаючи перевірку результатів).

СЕРЙОЗНІСТЬ ПОМИЛОК

Як і у випадку вартості тестів, існує два питання пов'язаних із серйозністю помилок: (1) їх вимір або оцінка для обчислення метрики $APFD_C$ для порядку тестів і (2) їх оцінка для використання інформації при пріоритезації тестів. Серйозність помилки пов'язана з витратами що відбудуться, якщо помилка залишилася у програмі після її релізу. Можливі різні підходи для виміру такої серйозності; наприклад:

1 можна виміряти серйозність помилки як суму грошей загублених у результаті збою викликаного даною помилкою (з урахуванням імовірності такого збою). Такий підхід може застосовуватися для ПЗ де індивідуальні збої можуть привести до катастрофічних наслідків, наприклад, судові позови, людські жертви, втрата обладнання.

2 можна оцінити вплив помилки на надійність ПЗ. Даний підхід застосовується для ПЗ де помилки можуть викликати лише незручність для користувачів, і малоймовірно що збій буде мати серйозні наслідки (наприклад, текстовий редактор). У цьому випадку, основним наслідком є зниження надійності ПЗ яке може привести до втрати клієнтів.

Подібно ситуації з вартістю тестів, нас цікавить як оцінка серйозності помилок до їхнього виявлення так і оцінка їх серйозності після виявлення. Коли тестування закінчене та у нас вже є інформація про помилки, можна оцінити їх серйозність для використання в метриці $APFD_C$ (хоча це не так просто як з вартістю тестів). З іншого боку, перед початком тестування, нам необхідно оцінити серйозність потенційних помилок для використання даної інформації в процесі пріоритезації. Тому нам необхідно зв'язати тести із серйозністю помилок які вони виявляють.

Якби ми знали які помилки виявляє кожний тест, і знали їхню серйозність, було б нескладно зв'язати тести із серйозністю помилок. Але на практиці, ця інформація недоступна до завершення тестування. Існують два підходи для такої оцінки:

1 оцінка критичності модулів; тут необхідно зв'язати серйозність помилки із критичністю модуля (або будь-якого іншого компонента програмного коду) у якому знаходиться ця помилка.

2 оцінка критичності тестів; тут необхідно зв'язати тести із серйозністю помилок які вони можуть виявити.

Оцінивши критичність модулів або тестів, необхідно інтегрувати інформацію про серйозність помилок у процес пріоритезації. В даній роботі, у дослідженнях ми застосуємо другий підхід. Критичність тестів використовується як метрика для оцінювання серйозності помилок які може виявити кожний тест. На практиці ми не можемо знати помилки наперед, тому необхідно оцінити критичність тестів не використовуючи інформацію про помилки які даний тест виявляє. У даній роботі, ми вважаємо що серйозність помилки визначається її впливом на надійність системи. Чим серйозніше помилка, тім частіше відбуваються пов'язані з нею збої. У даній роботі ми не диференціюємо збої, а враховуємо лише їх частоту.

ОПЕРАЦІЇ ТА ОПЕРАЦІЙНИЙ ПРОФІЛЬ

Для розуміння концепції операційний профіль, нам необхідно визначити що таке операція. Операцією є зовнішньо-ініційована дія виконана системою [23]. З погляду користувачів, операція може бути зрівняна з функцією що є зовнішньо ініційованою дією. Прикладами операцій можуть бути команди, транзакції та обробка зовнішніх подій.

Операційний профіль (operational profile) є множиною операцій та ймовірністю їх використання [23]; інакше кажучи, дана система S, операційний профіль для *S* – це список дій, які користувачі можуть робити з S, і частоту цих дій.

Розглянемо приклад текстового редактора. У цьому прикладі, операція «змінити шрифт» може виконуватися 20 % часу, операція «вирізати текст» – 10 %, «ввести текст» – 50 %, «копіювати текст» – 10 % і «вставити текст» – 10 %.

Операційний профіль часто використовується при розробці систем ПЗ, що включають дизайн, розробку, тестування та використання. Наприклад, одним застосуванням операційного профілю може бути тестування операцій пропорційно частоті їх використання. Наприклад, при

тестуванні текстового редактора наведеного в нашому прикладі, нам необхідно витратити половину зусиль тестуючи операцію «ввести текст».

У даній роботі ми використовували операційний профіль для оцінки серйозності помилок і критичності тестів. Для наших об'єктів дослідження, спочатку ми визначили безліч операцій застосовних до програм. Потім ми зв'язали операції з тестами. Це дозволило нам створити матрицю операцій для програм, де були спаровані операції з тестами. Матриця операцій використовується для визначення серйозності помилок і критичності тестів описаним пізніше методом.

КРИТИЧНІСТЬ ТЕСТІВ

Ми визначили критичність тесту як його важливість для користувачів програми. Критичність тестів може бути асоційована із частотою використання різних операцій у системі. Тест виконуючий операції які частіше використовуються більш критичний ніж тест виконуючий операції які рідко використовуються. Ми обчислили критичність кожного тесту як суму ймовірностей використання операцій виконаних тестом.

СЕРЙОЗНІСТЬ ПОМИЛОК

У даній роботі ми використовуємо операційний профіль для оцінки серйозності помилок. Ми вважаємо що збої у часто використовуваних операціях ведуть до більш частих збоїв при звичайному використанні програми, і такі збої можуть створювати більш значні проблеми користувачам. Тому таким збоям привласнюється більш висока серйозність ніж збоям пов'язаним з рідко використовуваними операціями. Для визначення серйозності помилок для наших об'єктів дослідження, ми використовуємо зв'язок між матрицею помилок (що несе інформацію які тести виявляють які помилки) і матриці операцій (що несе інформацію які тести виконують які операції). Ми спочатку визначаємо асоціацію між помилками та операціями. Цей зв'язок визначається матрицею помилок і матрицею операцій.

Повернемося до текстового редактору. Допустимо в програмі присутні три помилки. Нехай помилки 1 і 2 мають вплив на операції «змінити шрифт», «вирізати текст», «ввести текст»; помилка 3 має вплив на операції «копіювати текст» і «вставити текст». У цьому ж прикладі тести 1, 2 і 3 виконують операції «змінити шрифт», «вирізати текст» і «ввести текст», а тести 4-8 виконують операції «копіювати текст» і «вставити текст». Крім цього, помилка 1 виявляється тестами 1, 2 і 3; помилка 2 виявляється тестом 1; а помилка 3 виявляється тестами 4-8.

Таким чином, серйозністю помилки може бути сума ймовірностей використання операцій на які впливає ця помилка; наприклад, серйозність помилки 1 – це сума ймовірності операцій на які впливає помилка 1, та дорівнює 0,20 + 0,10 + 0,50 = 0,80 (з описаного вище прикладу). Серйозність помилки 2 дорівнює 0,80, а серйозність помилки 3 дорівнює 0,20.

Такий підхід дає помилкам 1 та 2 рівну серйозність. Помилка 1 виявляється тестами 1, 2 і 3, з іншого боку, помилка 2 виявляється тільки тестом 1. Якщо тести відповідають операційному профілю, то операції що виконуються більшою кількістю тестів повинні давати більший внесок при підрахунку серйозності помилки яка на них впливає.

Для вирішення даної проблеми ми враховуємо кількість тестів асоційованих з кожною операцією. Якщо на операцію впливають дві помилки, f_1 і f_2, і якщо f_1 виявляється більшою кількістю тестів ніж f_2, ми даємо більшу серйозність помилці f_1 ніж помилці f_2.

Одним методом для надання більш високої серйозності помилці 1 чим помилці 2 – це просто використовувати кількість тестів як вагу при обчисленні серйозності помилок. Нехай T_i буде список тестів асоційованих з операцією i. Нехай Op – буде множина m операцій пов'язаних з помилкою X, і нехай P_i буде ймовірністю асоційованої з операцією i. Серйозність помилки X може бути визначена такий чином $\sum_{i=1}^{m} P_i \times \|T_i\|$.

По цьому методі серйозність помилок 1, 2 і 3 у наших прикладах наступні:

серйозність помилки 1 дорівнює 3 * 0,20 + 3 * 0,10 + 3 * 0,50 = 2,40;

серйозність помилки 2 дорівнює 0,20 + 0,10 + 0,50 = 0,80;

серйозність помилки 3 дорівнює 5 * 0,10 + 5 * 0,10 = 1,0.

Однак цей метод робить помилку 3 більш серйозною, ніж помилку 2; це неправильно тому що помилка 2, на відміну від помилки 3, може впливати на найчастіше використовувану

операцію. Це суперечить нашому твердженню що помилки які зустрічаються в більш частіше використовуваних операціях більш серйозні ніж помилки які зустрічаються в менш частіше використовуваних операціях. Тому вводяться якісь значення бар'єру вище якого буде враховуватися кількість тестів і нижче якого – ні. Нехай P_H буде ймовірність використання найчастіше використовуваної операції на яку впливає дана помилка та P_L – ймовірність використання найменш часто використовуваної операції на яку впливає дана помилка. Значення бар'єру для T_H обчислюється формулою:

$$TH = \frac{P_H + P_L}{2}.$$

Таким чином, значення бар'єра для прикладу з текстовим редактором буде:

$$TH = \frac{0{,}50 + 0{,}10}{2} = 0{,}30.$$

Тому серйозність помилок при даному значенні бар'єра буде:
серйозність помилки 1 дорівнює 0,20 + 0,10 + 3 * 0,50 = 1,80;
серйозність помилки 2 дорівнює 0,20 + 0,10 + 0,50 = 0,80;
серйозність помилки 3 дорівнює 0,10 + 0,10 = 0,20.

ЕМПІРИЧНІ ДОСЛІДЖЕННЯ

Нами було проведене дослідження, ціль якого – вивчити, як різні розподіли та шкали вартості тестів і серйозності помилок можуть впливати на швидкість їх виявлення з точки зору метрики APFDC. Поняття критичності тесту застосовувалося для оцінки серйозності помилок у пріоритезації. Було розглянуто вплив різних розподілів та шкал вартості тестів, серйозності помилок і їх комбінацій на відносну ефективність методів пріоритезации.

ОБ'ЄКТИ ДОСЛІДЖЕННЯ

Як об'єкти даного дослідження використовувалися наступні програми:

- **empire** – мережева гра, складається з 63 014 – 64 396 строк коду, 10 версій, 1985 тестів та по 10 регресивних помилок в усіх версіях крім першої;
- **grep** – програма пошуку тексту, складається з 10 929 – 13 359 строк коду, 5 версій, 810 тестів та від 1 до 4 регресивних помилок в усіх версіях крім першої;
- **flex** – генератор лексичних аналізаторів, складається з 11 783 – 14 171 строк коду, 5 версій, 568 тестів та від 1 до 8 регресивних помилок в усіх версіях крім першої;
- **make** – програма що застосовується для побудови ПЗ, складається з 18 665 – 25 465 строк коду, 5 версій, 1044 тестів та від 3 до 5 регресивних помилок в усіх версіях крім першої;
- **sed** – потоковий редактор, складається з 8 063 – 11 911 строк коду, 3 версій, 1294 тестів та від 4 до 5 регресивних помилок в усіх версіях крім першої;
- **xearth** – програма що відображає картинку земної кулі, складається з 13 165 – 24 179 строк коду, 3 версій, 540 тестів та від 3 до 4 регресивних помилок в усіх версіях крім першої.

Використовуючи документацію для програми **empire**, були визначені 182 операції, використовуючи пряму асоціацію з командами. Кожний тест виконує послідовність команд, тому було легко асоціювати тести з операціями. Для інших програм, тести були згенеровані виходячи з мови специфікації тестів (test specification language (TSL)). Для кожної програми, була створена специфікація тестів базуючись на документації. Далі ці специфікації були перетворені в тестові фрейми (test frames) з яких були сгенеровані тести. Операції були створені для кожного вибору у файлі специфікації тестів; потім деякі операції із загальними характеристиками були об'єднані між собою. Далі, використовуючи інформацію про то як були сгенеровані тести, вони були зв'язані з операціями.

Для створення операційного профілю для програми **empire**, були задіяні добровольці які протягом деякого часу користувалися цією грою. Для інших програм, операційний профіль було сгенеровано випадковим чином.

МЕТОДИ ПРІОРИТЕЗАЦІЇ

У дослідженні були використані наступні методи пріоритезації:

- **fn-nofb** – пріоритезація по загальному покриттю функцій. Вставляючи в програму додатковий код, можна визначити для кожного тесту, які функції їм покриті. Ці тести можна пріоритезувати відповідно до кількості покритих ними функцій, сортуючи тести в порядку убування даного загального покриття.
- **fn-fb** – пріоритезація по додатковому покриттю функцій. Пріоритезація по загальному покриттю функцій планує тести в порядку досягнення сукупного покриття функцій. Однак, виконавши тест і покривши деякі функції, доцільно, при наступному тестуванні покрити ще непокриті функції. Пріоритезація по додатковому покриттю функцій вибирає тест, що досяг найбільшого покриття ще не покритих функцій, повторюючи цей процес доти, поки жоден з тестів, що залишилися, не зможе збільшити покриття функцій. Коли це відбувається, даний алгоритм застосовується рекурсивно до тестів, що залишилися.
- **diff-fn-nofb** – пріоритезація по загальному покриттю змінених функцій. Цей метод схожий на fn-nofb, але розглядає тільки нові або змінені функції. При рівній кількості покритих змінених функцій, враховується кількість усіх покритих функцій.
- **diff-fn-fb** – пріоритезація по додатковому покриттю змінених функцій подібна пріоритезации по додатковому покриттю функцій, за винятком того, що враховуються тільки змінені або нові функції. При рівній кількості покритих змінених функцій, враховується кількість усіх покритих функцій.
- random упорядковує тести випадковим образом.

Для того щоб врахувати вартість і критичність тестів у методах пріоритезации, було знайдене відношення критичності тесту до його вартості. У данім дослідженні ми розглянули три методи впровадження цього відношення в методи пріоритезации:

- **Af** – тести пріоритезуються відповідно до покриття функцій (з або без урахуванням модифікації програмного коду), відношення критичності тесту до його вартості має другорядне значення і враховується при рівнозначності покриття.
- **Mult** – тести пріоритезуються відповідно до добутку покриття функцій (з або без урахуванням модифікацій програмного коду) і відношення критичності тесту до його вартості; у цьому випадку внесок покриття та вартості із критичністю рівнозначні.
- **Rf** – тести пріоритезуються відповідно до відношення критичності тесту до його вартості, а покриття функцій (з або без урахуванням модифікацій програмного коду) має другорядне значення і ураховується при рівнозначному відношенні критичності до вартості.

ШКАЛИ ВАРТОСТІ ТЕСТІВ

У дослідженні були розглянуті чотири шкали вартості тестів.

- **Unit**: дана шкала не розрізняє вартість тестів, у цьому випадку вартість тесту не враховується;
- **Orig**: дана шкала привласнює кожному тесту його вартість (справжню або штучно сгенеровану) отриману описаними раніше методами;
- **Buck-lin**: у даній шкалі, значення вартості тестів (взяті зі шкали orig) відсортовані й рівномірно розподілені в гнізда. Вартість тестів відповідає номеру гнізда починаючи з одиниці;
- **Buck-exp**: дана шкала подібна buck-lin, крім того що вартість тесту є не номер гнізда n, а $b^n - 1$, де $b = 2$ та n – номер гнізда. Дана шкала – альтернатива попередньої, яка збільшує значимість найбільш дорогих тестів.

ШКАЛИ КРИТИЧНОСТІ ТЕСТІВ І СЕРЙОЗНОСТІ ПОМИЛОК

У дослідженні були розглянуто чотири шкали критичності тестів і серйозності помилок.

- **Unit**: дана шкала не розрізняє між критичністю тестів і серйозністю помилок, у цьому випадку критичність тесту та серйозність помилок не враховується;
- **Orig**: дана шкала привласнює кожному тесту його критичність і кожній помилці її серйозність отримані описаними раніше методами;
- **Buck-lin**: у даній шкалі, значення критичності тестів і серйозності помилок (взяті зі шкали orig) відсортовані та рівномірно розподілені в гнізда. Критичність тестів і серйозність помилок обчислюється по номеру гнізда;
- **Buck-exp**: дана шкала подібна buck-lin, тільки замість номера гнізда n, використовується $b^n - 1$, де $b = 2$ та n – номер гнізда. Дана шкала – альтернатива попередньої, яка збільшує значимість найбільш серйозних помилок і відповідних їм тестів.

РЕЗУЛЬТАТИ

Дослідження складалося із трьох частин. У першій частині ми розглянули вплив трьох методів впровадження інформації про вартість і критичності тестів у процес пріоритезации. У другій частині, ми розглянули як впливає вибір шкали вартості тестів і серйозності помилок на відносну ефективність методів пріоритезации. У третій частині ми розглянули для кожного методу пріоритезации відносний вплив усіх 16 комбінацій шкал на швидкість виявлення помилок.

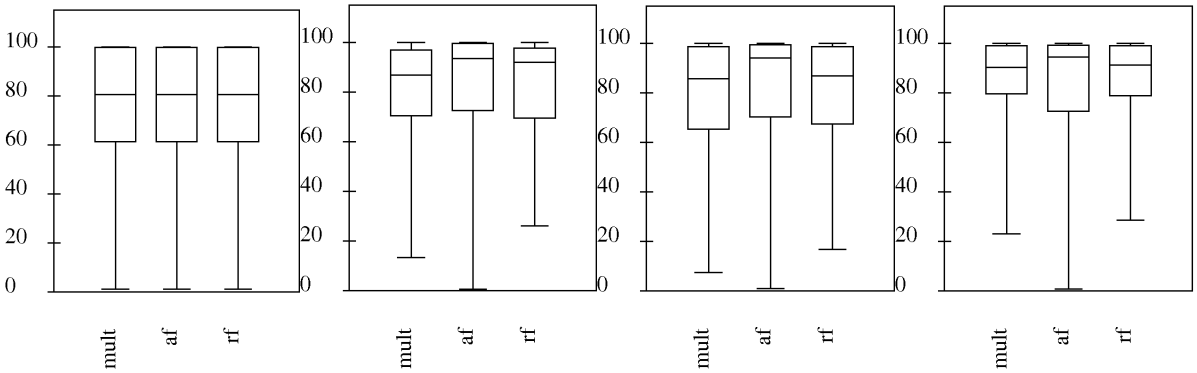

Рис. 3. Метод fn-nofb зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp.

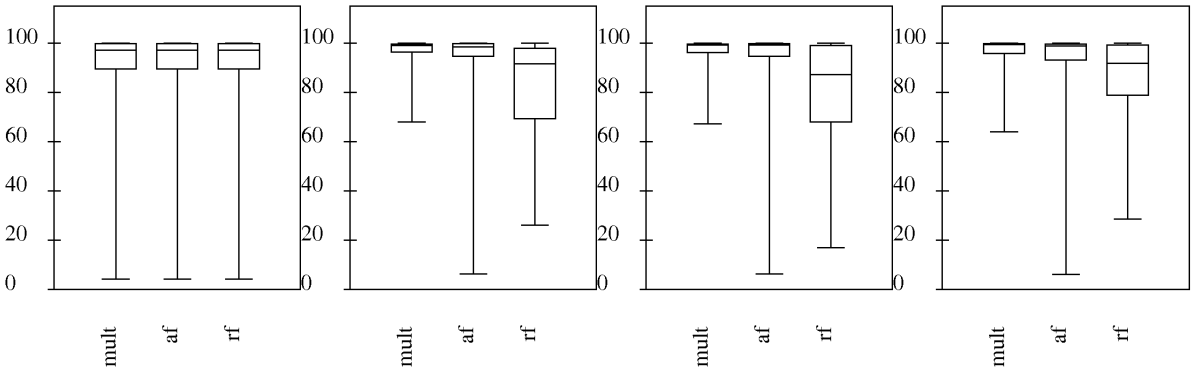

Рис. 4. Метод fn-fb зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

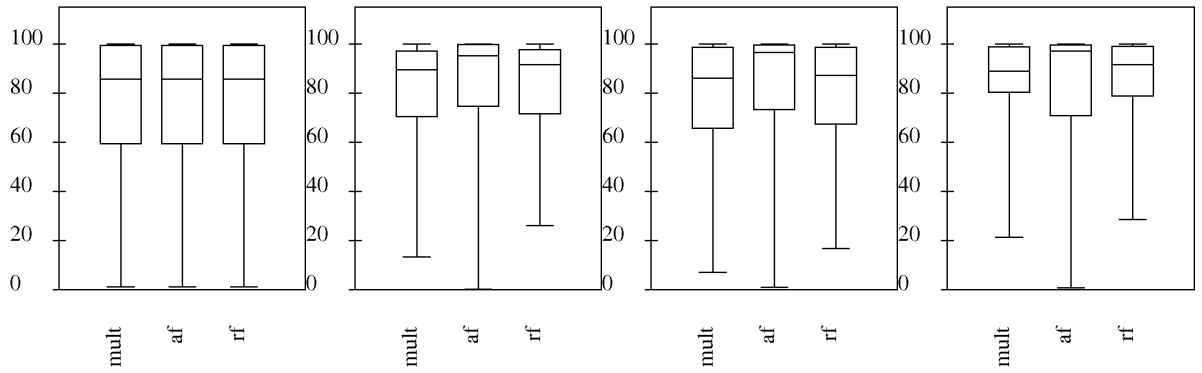

Рис. 5. Метод diff-fn-nofb зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

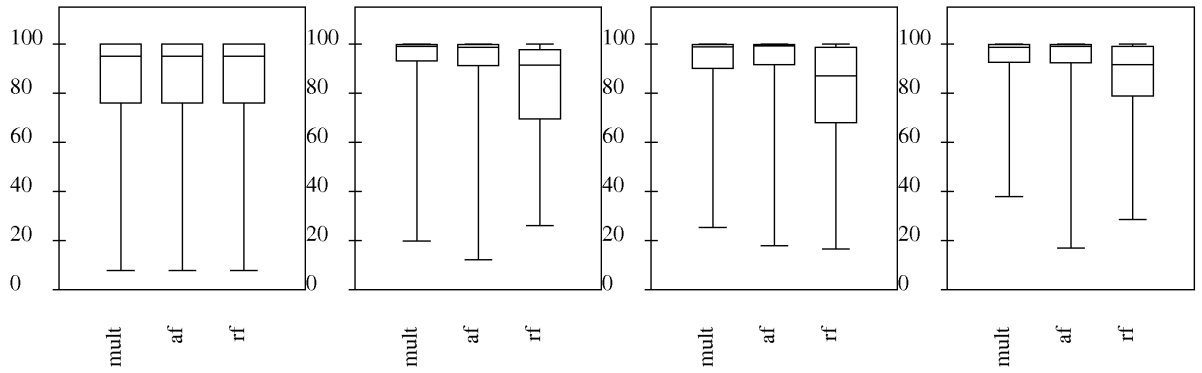

Рис. 6. Метод diff-fn-fb зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

Дослідження 1. У даному дослідженні ми розглянемо три методи комбінації інформації (**af**, **mult** та **rf**) про вартість і критичність тестів з інформацією про покриття елементів програмного коду.

У даному дослідженні були отримані дані використовуючи різні розподіли (шкали) вартості тестів і серйозності помилок (**orig/orig, buck-lin/buck-lin, buck-exp/buck-exp**) для кожного з п'яти методів пріоритезації й трьох видів комбінації інформації. Був проведений статистичний аналіз Анова (Anova). Результати представлені у вигляді діаграм розмаху на *Рис. 1 – Рис. 6*.

Як ми бачимо із графіків на *Рис. 1 – Рис. 6*, для методів **fn-nofb** та **diff-fn-nofb**, у трьох розглянутих шкалах вартості тестів і серйозності помилок, **af** мав найкращі результати; а для методів **fn-fb** та **diff-fn-nofb**, **af** і **mult** були значно краще ніж **rf**. Тобто, для методів без зворотного зв'язку (по загальному покриттю функцій), комбінація **af** була найефективніша, а для методів зі зворотним зв'язком (по додатковому покриттю функцій), як **af** так і **mult** є ефективними. Анова аналіз показав що не всі результати є статистично значимими (для $\alpha = 0{,}05$). Значимими являються: метод **fn-nofb** зі шкалою **exp/exp** ($p = 0{,}0234$), метод **fn-fb** зі шкалами **orig/orig**, **buck-lin/buck-lin** та **buck-exp/buck-exp** ($p = 0$), а також метод **diff-fn-fb** зі шкалами **orig/orig** ($p = 2{,}2622*10^{-7}$), **buck-lin/buck-lin** ($p = 1{,}926310^{-11}$) та **buck-exp/buck-exp** ($p = 3{,}4632*10^{-4}$).

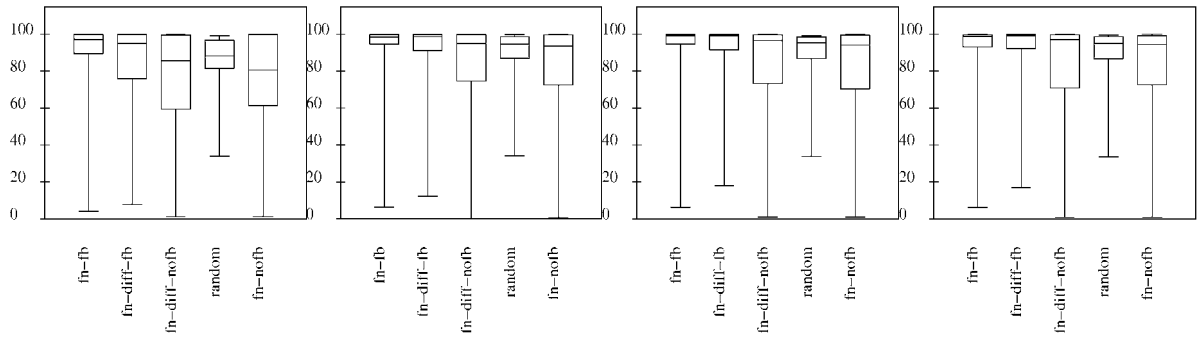

Рис. 7. Af комбінація зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

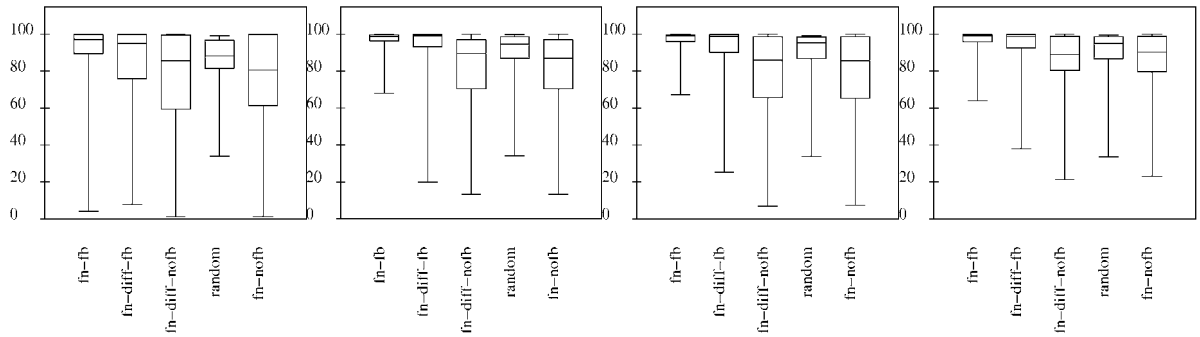

Рис. 8. Mult комбінація зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

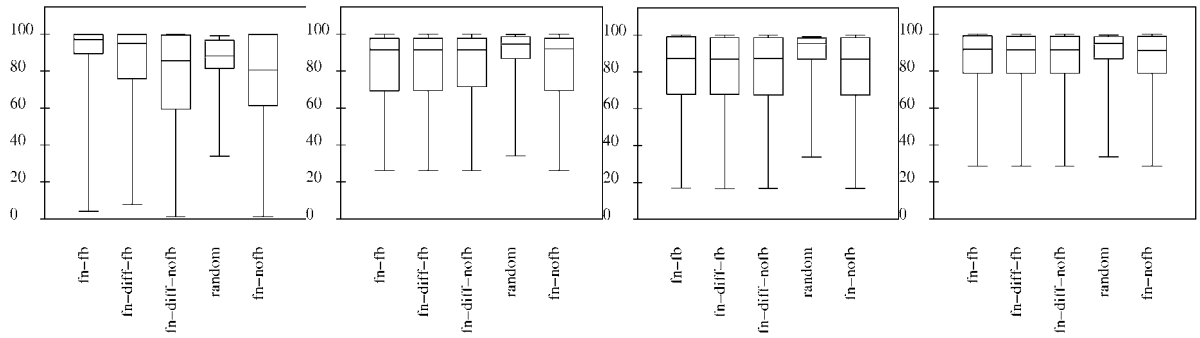

Рис. 9. Rf комбінація зі шкалами, зліва направо: unit/unit, orig/orig, buck-lin/buck-lin та buck-exp/buck-exp

Дослідження 2. У даному дослідженні був вивчений вплив шкали вартості тестів і серйозності помилок на відносну ефективність методів пріоритезації. Для кожного із трьох методів комбінації інформації і чотирьох комбінацій шкал (**unit/unit, orig/orig, buck-lin/buck-lin, buck-exp/buck-exp**), ми зрівняли між собою п'ять методів пріоритезации. Подібно попередньому дослідженню, проведемо статистичний аналіз Анова. У кожному із дванадцяти випадків Анова аналіз показав статистичну значимість відмінностей між методами пріоритезации. Найменше значення $p = 0{,}0263$ було досягнуто комбінацією **rf** та шкалою **buck-exp/buck-exp**.

Як ми бачимо з графіків на *Рис. 7 – Рис. 9* (представлених діаграмами розмаху), з погляду середнього значення метрики $APFD_C$, при комбінації **af** шкала має незначний вплив на відносну ефективність методів пріоритезации. За винятком шкали **orig/orig**, методи пріоритезации в порядку убування ефективності такі: **fn-fb, fn-diff-fb, random, fn-diff-nofb, fn-nofb**. Також, з точки зору середнього значення метрики $APFD_C$, при комбінації **mult** шкала зовсім не має

впливу на відносну ефективність методів пріоритезации. З погляду середнього значення метрики $APFD_C$, при комбінації **rf**, шкала має більш значний вплив на відносну ефективність методів пріоритезації. При даній комбінації методи пріоритезации мають незначну відмінність у їхній ефективності; набагато більший внесок у їхню ефективність має шкала. У шкалах **orig/orig, buck-lin/buck-lin** та **buck-exp/buck-exp**, метод **random** був найкращим, що означає марність пріоритезації в даних випадках. З іншого боку, у шкалі **unit/unit** вартість фактично не враховується і відносна ефективність методів пріоритезація така ж як і в інших комбінаціях.

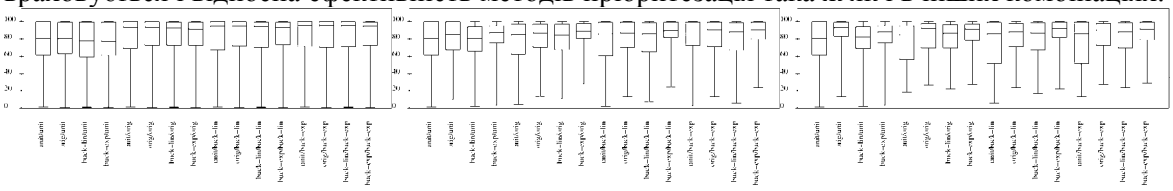

Рис. 10. **Метод fn-nofb з комбінаціями, зліва направо: af, mult та rf**

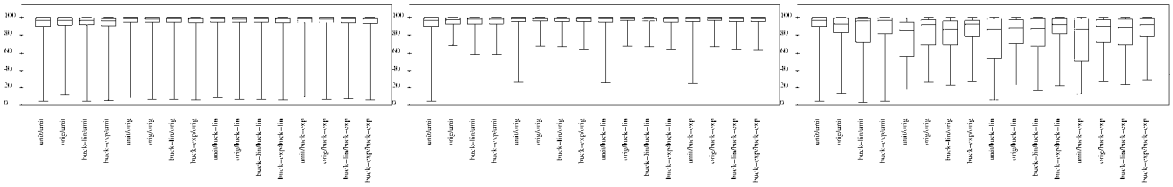

Рис. 11. Метод fn-fb з комбінаціями, зліва направо: af, mult та rf

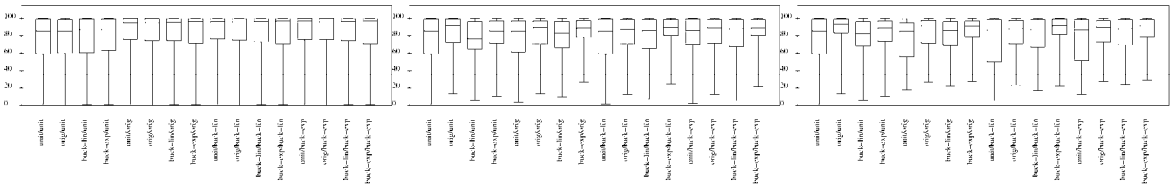

Рис. 12. Метод diff-fn-nofb з комбінаціями, зліва направо: af, mult та rf.

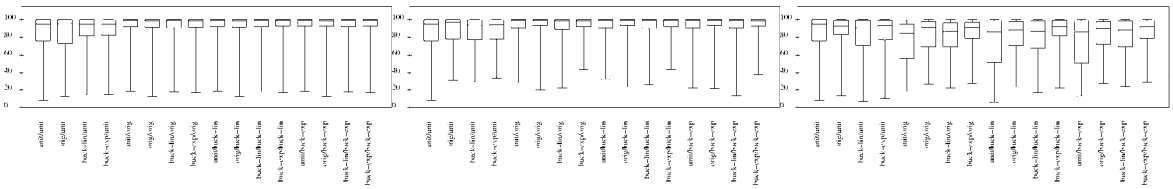

Рис. 13. Метод diff-fn-fb з комбінаціями, зліва направо: at, mult та rf

Дослідження 3. У даному дослідженні ми розглянули для чотирьох методів пріоритезації ітрьох методів комбінації, усі 16 шкал. Для методу пріоритезації **fn-fb** і комбінації **af**, відмінності в середніх значеннях $APFD_C$ не є статистично значимими ($p = 0,4313$). В усіх інших одинадцяти випадках, відмінності були статистично значимі (максимальне значення $p = 0,0015$).

Як ми бачимо з графіків на *Рис. 10 – Рис. 13* (представлених діаграмами розмаху), для метода **fn-nofb**, розмах середніх значень $APFD_C$ для різних шкал максимальний для **rf** – від 74,92 (**unit/orig**) до 86,55 (**orig/unit**); потім іде **mult** – від 75,81 (**unit/unit**) до 85,84 (**buck-exp/buck-exp**); та найменшій розмах для **af** – від 74,54 (**buck-exp/unit**) до 81,82 (**unit/buck-exp**). Для метода **fn-fb**, розмах середніх значень невеликий при комбінаціях **af** та **mult** (менш ніж 5), але є значним для **rf** – від 75,04 (**unit/orig**) до 92,71 (**unit/unit**). Для метода **diff-fn-nofb**, розмах є максимальний при **rf** – від 74,94 (**unit/orig**) до 86,85 (**orig/unit**), потім іде **mult** – від 76,07 (**buck-lin/unit**) до 85,62 (**buck-exp/buck-exp**); та найменший розмах при **af** – від 76,20 (**orig/unit**) до 84,15 (**buck-exp/buck-exp**). Для метода **diff-fn-fb**, розмах є максимальним при **rf** – від 74,93 (**unit/orig**) до **orig/unit** (86,67); потім іде **af** – від 85,01 (**orig/unit**) до 91,29 (**buck-exp/buck-lin**); найменший розмах має місце при **mult** – від 86,19 (**unit/unit**) до 92,62 (**buck-exp/buck-lin**).

ВИСНОВКИ

У даній роботі була розглянута пріоритезація, яка враховує вартість тестів і серйозність помилок. На відміну від попередніх досліджень, серйозність помилок була заснована на їхньому впливі на надійність програми. Декілька програм було використано як об'єкти дослідження. В одній з них були використані, як справжні дані про вартість тестів, так і справжній операційний профіль. Також як вартість тестів, так і серйозність помилок були досліджені в чотирьох різних шкалах і вивчений їхній вплив на ефективність пріоритезации. Було розглянуто три підходи до впровадження інформації про вартість і критичність тестів у процес пріоритезації та досліджена їх ефективність.

Вибір шкали має значний вплив на ефективність пріоритезации. Важливо те, що вибір кращого методу пріоритезації залежить від шкали вартості тестів і серйозності помилок. Різні шкали неоднаково впливають на різні методи пріоритезації. Тому вибір методу пріоритезації повинен враховувати шкалу. Також, методи комбінації **af** і **mult**, а особливо **af**, дають найкращі результати.

Автор дякує Г. Ротермела, С. Элбаума та Д. Рутруфа за участь у проведенні описаних досліджень.

ЛІТЕРАТУРА:

1. Ghezzi C., Jazayeri M., Mandrioli D. Fundamentals of Software Engineering. – Upper Saddle River: Prentice Hall, 1991. – 573 p.
2. Avritzer A., Weyuker E.J. The automatic generation of load test suites and the assessment of the resulting software // IEEE Transactions on Software Engineering. – 1995. – 21, № 9. – P. 705-716.
3. Wong W., Horgan J., London S., Agrawal H. A study of effective regression testing in practice // In Proceedings of the Eighth International Symposium on Software Reliability Engineering. – Albuquerque, NM, USA. – 1997. – P. 230-238.
4. Rothermel G., Untch R., Chu C., Harrold M.J. Test case prioritization: an empirical study // In Proceedings of the International Conference on Software Maintenance. – Oxford, England, UK. – 1999. – P. 179-188.
5. Elbaum S., Malishevsky A., Rothermel G. Prioritizing test cases for regression testing // In Proceedings of the International Symposium on Software Testing and Analysis. – Portland, Oregon. – 2000. – P. 102-112.
6. Elbaum S., Malishevsky A., Rothermel G. Incorporating Varying Test Costs and Fault Severities into Test Case Prioritization // Technical Report 00-60-09. – Computer Science Department. – Oregon State University. – August 2000. – 14 p.
7. Rothermel G., Untch R.H., Chu C., Harrold M.J. Test case prioritization // IEEE Transactions on Software Engineering. – 2001. – 27, № 10. – P. 92-948.
8. Jones J.A., Harrold M.J. Test-suite reduction and prioritization for modified condition/decision coverage // In Proceedings of the International Conference on Software Maintenance. – Florence, Italy. – 2001. – P. 92–101.
9. Elbaum S., Malishevsky A., Rothermel G. Incorporating varying test costs and fault severities into test case prioritization // In Proceedings of the 23rd International Conference on Software Engineering. – Toronto, Ontario, Canada. – May 2001. – P. 329-338.
10. Elbaum S., Malishevsky A. G., Rothermel G. Test Case Prioritization: A family of empirical studies // IEEE Transactions On Software Engineering. – 2002. – 28, № 2. – P. 159-182.
11. Srivastava A., Thiagarajan J. Effectively prioritizing tests in development environment // In Proceedings of the International Symposium on Software Testing and Analysis. – Via di Ripetta, Rome – Italy. – 2002. – P. 97-106.
12. Kim J.-M., Porter A. A history-based test prioritization technique for regression testing in resource constrained environments // In Proceedings of the International Conference on Software Engineering. – Orlando, Florida, USA. – 2002. – P. 119-129.
13. Malishevsky A. G. Test case prioritization // Ph.D. Dissertation. – Oregon State University, Corvallis, Oregon, USA. – 2003. – 291 p.
14. Do H., Rothermel G., and Kinneer A. Empirical Studies of Test Case Prioritization in a JUnit Testing Environment // Proceedings of the International Symposium on Software Reliability Engineering. – Saint-Malo, Bretagne, France. – November 2004. – P. 113-124.
15. Srikanth H. Value-driven system level test case prioritization // Ph.D. Dissertation – North Carolina State University, Releigh, NC. – 2005. – 92 p.
16. Korel B., Tahat L. H., and Harman M. Test Prioritization Using System Models. // In Proceedings of the 21st IEEE International Conference on Software Maintenance. – Budapest, Hungary. – September 2005. – P. 559-568.
17. Do H. and Rothermel G. A Controlled Experiment Assessing Test Case Prioritization Techniques via Mutation Faults // Proceedings of the IEEE International Conference on Software Maintenance. – Budapest, Hungary. – September 2005. – P. 411-420.
18. Do H., Rothermel G., and Kinneer A. Prioritizing JUnit test cases: An empirical assessment and cost-benefits analysis // Empirical Software Engineering: An International Journal. – March 2006. – 11, № 1. – P. 33-70.

19. Do H. and Rothermel G. On the use of mutation faults in empirical assessments of test case prioritization techniques // IEEE Transactions on Software Engineering. – September 2006. – 32, № 9. – P. 733-752.
20. Малышевский А.Г. Пріоритезація тестов в регрессивном тестировании // Системні дослідження та інформаційні технології. – 2006. – № 4. – С. 16-32.
21. Malishevsky A. G., Ruthruff J., Rothermel G., and Elbaum S. Cost-cognizant test case prioritization // Technical Report TR-UNL-CSE-2006-0004. – Department of Computer Science and Engineering. – University of Nebraska – Lincoln. – March 2006. – 41 p.
22. Малышевский А.Г. / Использование информации о стоимости тестов и серьезности ошибок в процессе пріоритезации тестов // Системні дослідження та інформаційні технології. – 2008. – № 1. – С. 63-78.
23. Musa J. Software Reliability Engineering. – New York, NY: McGraw-Hill, 1999. – 391 p.

Рецензенти: д.т.н., проф. Фісун М.Т.
д.т.н., проф. Кондратенко Ю.П.

Стаття надійшла до редколегії 24.01.09