УДК 004.052.42:004.85 **DOI:** 10.31673/2412-9070.2024.050854
А. П. САМОЙЛЕНКО, аспірант,
ORCID: 0009-0004-6353-1877
Державний університет інформаційно-комунікаційних технологій, Київ

МЕТОДИ ПРОГНОЗУВАННЯ РЕЗУЛЬТАТІВ МУТАЦІЙНОГО ТЕСТУВАННЯ

Мутаційне тестування є широко визнаною технікою для оцінки ефективності наборів тестів у виявленні помилок у програмному забезпеченні. Незважаючи на ефективність цього методу, виконання мутацій вимагає значних обчислювальних витрат, що ускладнює його широке застосування. Для вирішення проблеми ефективності дослідники пропонують різні підходи, такі як селективне, ослаблене і прогнозоване мутаційне тестування. Це зумовлено використанням мутаційного тестування як метрики для оцінки якості набору тестів, розробленого за допомогою фазового тестування. Як мутаційне тестування, так і фаззінг тестування є обчислювально інтенсивними процесами, створюючи значну перешкоду, яка потребує інноваційних рішень. У зв'язку з цим, оптимізація обчислювальних ресурсів стає ключовою областю уваги, метою якої є збалансування ефективності без компромісу щодо точності та глибини використаних методик тестування. Ця складна взаємодія між збалансуванням витрат, обчислювальною потужністю і ефективністю тестування підкреслює потребу у стратегічних методах оптимізації, призначених для динамічних і вимогливих до ресурсів сценаріїв оцінки. Пошук підходів, які дозволять швидко та точно оцінити ефективність мутаційного тестування, залишається відкритою проблемою, оскільки методи зменшення мутацій не переважають випадковий вибір мутантів.

Одним із шляхів вирішення такої проблеми є використання методів прогнозування, що дозволяють передбачити результати, не виконуючи мутації. Дані методи прогнозування мутаційного тестування представляють собою новий напрямок досліджень в області методів оптимізації мутаційного тестування. Розглянуті методи можна класифікувати за типом машинного навчання (навчання з учителем та без вчителя), за типами ознак (динамічні та статичні), за рівнем прогнозування (на рівні мутантів та на рівні тестового набору), за типом мутацій, що використовуються (мутації першого порядку та мутації вищого порядку).

Ключові слова: мутаційне тестування, машинне навчання, глибоке навчання, модель, оптимізація, фреймворк, аналіз даних.

Вступ

Зниження впливу людського фактору під час тестування є важливим чинником для підвищення якості програмного забезпечення. Один із способів досягнення цієї мети - використання методу фаззінгу, який полягає у генерації випадкових вхідних послідовностей. Щоб оцінити ефективність цих вводів, найчастіше використовується метрика покриття коду. Однак важливо відзначити, що ця метрика, хоч і має високий ступінь кореляції з можливістю виявлення порушення працездатного функціонування ПЗ, але має обмежену здатність виявлення саме відмов.

Інструментом, що широко використовується для оцінки якості тестових наборів є мутаційне тестування. Але основною проблемою, як фаззінгу, так і мутаційного тестування є висока обчислювальна вартість. Це у свою чергу підіймає ряд задач, які необхідно вирішити, щоб ефективно застосовувати разом ці методики.

Єдиним фреймворком що реалізує фаззінг з мутаційним тестуванням є Mu2 [1]. Як показано у дослідженні, застосування до фаззінгу такої метрики як мутаційне тестування, може створювати тестовий набір вищої якості у порівнянні з покриттям коду [2]. В іншому дослідженні [3] аналогічного підходу автори зробили висновок, що експериментальні результати свідчать про

здатність мутаційного тестування мати перевагу, як у відношенні до покриття коду, так і у виявленні помилок. Застосування мутаційного тестування для оцінки фаззінгу [4] показало, що такі популярні фаззери, як AFL та інші, мають малу здатність до виявлення мутантів, а оскільки мутанти є коректною заміною реальних відмов, з цього можна зробити висновок, що запропонований підхід має перевагу у порівнянні з іншими методами [5].

Однак, для Mu2 - однією з подальших майбутніх напрямків дослідження є оптимізація мутаційного тестування, оскільки такий метод фаззінгу потребує досить агресивних оптимізацій [2]. Ситуацію ускладнює також те, що при застосуванні цих методик постає ряд задач, одна з яких - це знаходження необхідних оптимізацій [1]. Також, зважаючи на новітність даної проблеми і недостатність досліджень, крім можливих технічних причин, виділяють проблему привернення науковців до вивчення застосування фаззінгу з мутаційним тестуванням.

Більшість технік оптимізації ґрунтуються на використанні статичних тестових наборів, проте такі підходи виявляються непридатними для фаззінгу через використання випадкових вхідних послідовностей для різних мутантів. Внаслідок цього традиційні методи оптимізації стають неефективними у контексті фазерів, тому основною задачею, що необхідно вирішити для ефективного застосування фазерів з мутаційним тестуванням [6] - це знаходження оптимізацій, які не ґрунтуються на статичних тестових наборах.

Техніки оптимізації мутаційного тестування базуються на твердженні, що для оцінки ефективності мутантів кожен з мутантів має бути виконаний. З цього з'являється питання чи можна отримати результати мутаційного тестування без виконання мутантів? Один з розв'язків - застосування методів прогнозування, які дозволять отримувати результати, які є компромісом між точністю і часом обчислення.

Аналіз останніх досліджень і публікацій

Методи прогнозування результатів мутаційного тестування дозволяють отримувати міру якості тестового набору без виконання мутантів. За класифікацією [7] можна виділити три види таких методів:

1. Прогнозування на рівні коду системи, що тестується.
2. Прогнозування на рівні тестів.
3. Прогнозування на рівні мутантів.

Зважаючи на новітність даних методів, огляди досліджень існують тільки як частина статей, мета яких - описати новий підхід.

Результати дослідження

Метою дослідження є аналіз та представлення огляду моделей і методів прогнозування результатів мутаційного тестування та виявлення нових перспективних напрямків досліджень.

Машинне навчання використовується у мутаційному тестуванні для прогнозування результатів тестування без фактичного виконання всіх можливих мутацій. Основна ідея полягає в тому, щоб побудувати модель, яка може передбачати чи буде конкретний мутант "знищений" тестовим набором, чи залишиться "живим", і це робиться на основі різних ознак і характеристик.

Під час відбору ознак у контексті аналізу даних та машинного навчання розглядається завдання вибору підмножини ознак або змінних із загального набору даних для покращення якості моделі. Головна мета — відібрати найбільш інформативні та значущі ознаки для врахування у моделі, уникаючи зайвої або корельованої інформації, що може призвести, наприклад, до перенавчання моделі. Цей етап відбору ознак є ключовим у підготовці даних для ефективного та точного моделювання.

Загалом можна виділити кілька категорій ознак у контексті коду. Динамічні ознаки є тими, що збираються під час виконання коду, тоді як статичні ознаки визначаються характеристиками, які можна зібрати з вихідних текстів програми. Динамічні ознаки відображають поведінку коду під час його виконання, в той час як статичні ознаки вказують на характеристики програмного коду, які залишаються незмінними без його активного виконання. Окрім цього

існує категорія ознак, пов'язаних з історією, яка включає в себе аспекти обслуговування коду та його еволюцію в часі. Ця категорія ознак дозволяє враховувати аспекти історичного контексту при аналізі програмного коду.

Машинне навчання використовується в PMT (Predictive Mutation Testing) [8] для передбачення результатів мутаційного тестування без фактичного виконання конкретної мутації. Цей метод визначається через використання моделей машинного навчання, які навчаються на підготовлених даних, базуючись на розподілених властивостях відповідно до техніки розповсюдження, інфікування, виконання (PIE).

Головна ідея застосування цієї техніки - виявити місця у коді, що тестується, з помилками, які не були виявлені тестовим набором. PIE вказує на три категорії, які можуть бути використані як умови, які повинні бути задоволені для того, щоб визначити чи мутант був уражений тестом. Абревіатура PIE визначається наступним чином:

1. Execution (виконання): змінений оператор коду повинен бути виконаний тестом.

2. Infection (інфікування): стан програми повинен бути змінений безпосередньо після виконання зміненого оператора коду так, щоб тест міг відрізнити мутанта від оригінальної програми.

3. Propagation (розповсюдження): змінений стан програми повинен розповсюджуватися до виведення тесту так, щоб виведення тесту для мутанта відрізнялося від виведення тесту для оригінальної програми.

У дослідженні [8] використовувалося 14 властивостей, з яких найефективнішими виявилися динамічні властивості, зокрема ті, що пов'язані з інформацією про покриття коду. За результатами аналізу цих властивостей та інших можна зробити висновок, що більша кількість виконань призводить до вищого рівня передбачуваності. Тому у контексті фаззерів особливо важливо провести аналіз продуктивності цього методу, оскільки одним із можливих застосувань мутаційного тестування в якості засобу зворотного зв'язку є використання його на області тестів, які мають найбільше покриття програмного коду [6]. Також важливо зауважити, що цей метод не може бути застосований без використання властивостей тестів і базується не тільки на характеристиках програмного коду.

Однак, попередні дослідники не розглядали вплив таких мутантів, які не виконувались ні одним тестом із тестового набору - тобто непокритих мутантів. У дослідженні [9] було показано, що показники точності прогнозування суттєво зменшуються при використанні для тренування моделі тільки покритих мутантів. Однак автори запропонували також і підхід, що покращує ефективність даного методу. Покращення стало можливим завдяки застосуванню модифікованої моделі машинного навчання, а також формуванню іншої множини ознак. Формування нової множини ознак і порівняння з множиною ознак з дослідження [8] відбулося внаслідок обчислення ступеня значущість ознаки. В результаті нова множина у порівнянні сформувалася в більшості внаслідок складніших ознак (наприклад, таких як метрика Холстеда), ознак, які є статистично значущі та на противагу іншим дослідженням, що статичні ознаки відіграють ключову роль в новому підході. В іншій роботі [10] автори відзначають, що тренування моделі тільки на покритих мутантах спричиняє ріст продуктивності і змінює порядок важливості ознак - наприклад, найбільш вагома ознака, що позначає скільки разів мутований рядок коду програми був виконаний тестом втратила свою значимість - ці ж результати є ідентичними до результатів дослідження [9].

Ще однією загрозою корректності даного методу є застосування різних інструментів для мутаційного тестування, а також використання методу за допомогою імплементації на інших мовах програмування. Використання іншої мови програмування може зумовлювати інший набір мутаційних операторів, що призводить до інших результатів і як наслідок - до інших показників ефективності. Дане питання розглядається у роботі [10] - автори відворили оригінальне дослідження [8] і імплементували також PMT на мові програмування C. В результаті дослідження [10] було виявлено:

1. Значущість ознак для різних мов дуже розрізняється - наприклад, найважливішою ознакою на C стала кількість перевірок у тестовому класі.

2. Результати між проєктами на Java та C є порівняними при тестуванні за однакових умов.

CBUA [7] - це підхід на основі покриття без нагляду (Coverage-Based Unsupervised Approach), що використовує дані про покриття коду для оцінки вірогідності виживання мутантів. Ця оцінка вираховується за допомогою ймовірностної формули, що виражає процес мутаційного тестування як випробування Бернуллі. Тобто, коли мутований вираз є таким, що покривається тестом з тестового набору, відповідний мутант має деяку вірогідність виживання.

CBUA використовує навчання без вчителя. В інших дослідженнях розглядається тільки ефективність натренованої моделі, але не враховується фактор високої вартості у процесі навчання, яке в результаті буде все одно залежати від високої вартості мутаційного тестування. Оскільки такі моделі, як Random Forest реалізують навчання з вчителем, то вони потребують великої кількості розмічених даних. Варто також зазначити, що моделі, що використовують навчання з учителем використовують припущення, що дані для тренування та дані для валідації, тестування моделі мають однаковий розподіл. Але таке припущення може бути неправильним, що в результаті призводить до більшої помилки при прогнозуванні. Іншою перевагою даного методу є те, що при додаванні нових тестових випадків не зменшується ефективність мутаційного тестування, що може порушуватись при використанні навчання з вчителем.

Аналогічно PMT, CBUA також має здатність визначати потенційно живих мутантів, що допомагає розробникам виявляти слабкі місця у поточному тестовому наборі та додавати нові тестові випадки відповідно. Обидва методи виявляються ефективними у виявленні мутантів, які покриті, але не знищені поточним тестовим набором. Зокрема, CBUA вирізняється своєю конкурентоспроможністю з PMT при наявності обширних даних для навчання, переважаючи його в ситуаціях з обмеженими навчальними даними. Крім того, CBUA вирізняється високою універсальністю, демонструючи високу ефективність у різних умовах застосування мутаційного тестування, тобто показує високу ефективність на різних інструментах мутаційного тестування, різних типах тестування та за рахунок простої реалізації даний підхід може використовуватися як базовий для порівняння нових підходів.

Методи PMT і CBUA належать до категорії методів прогнозування, де ефективність мутаційного тестування оцінюється на рівні мутанта за допомогою прогнозування результатів. У контексті дослідження, яке представлене у роботі [11], автори пропонують використання машинного навчання для визначення ефективних та неефективних тестів з тестового набору, а використання тільки статичних ознак дозволяє оцінити цю ефективність не виконуючи тестів. Для запропонованого методу застосовується зовсім інший підхід, де ефективність мутаційного тестування визначається на основі оцінки ефективності тестів - наприклад, за допомогою метрик якості чи забруднювачів коду або тестів. У цьому дослідженні прогнозування відношення вбитих мутантів до живих формулюється як проблема двокласової класифікації. У подальшому дослідженні [12] було показано, що використання більш легковісних ознак є кращим як для точності прогнозування, так і для застосування у реальних умовах.

Метод, що оцінює ефективність мутаційного тестування на рівні коду системи, що тестується, запропонований у дослідженні [13]. У цьому підході проблема прогнозування виражається як задача багатокласової класифікації, де розрізняють наступні класи - низької, середньої та високої ефективності мутаційного тестування. Прогнозування ґрунтується на метриках як коду системи, що тестується, так і тестового набору. Однак недоліком даної системи може бути те, що дане дослідження проводилося тільки використовуючи один проєкт, на якому як і відбувалося тренування моделі, так і застосування методу прогнозування.

Такі методи прогнозування мутаційного тестування як PMT можуть бути спроєктованими лише для обчислення ефективності мутацій для об'єктно-орієнтованих програм. Щоб подолати недоліки наявних методів, був запропонований метод [14] для прямого прогнозування ефективності мутаційного тестування для кожного виразу у процесно-орієнтованих програмах. В основу методу покладена математична модель, що базується на припущеннях щодо впливу помилки на програму.

Мутаційне тестування з використанням мутацій вищого порядку - це техніка, що дозволяє уникнути високих витрат на обчислення шляхом зменшення кількості мутантів та уникнути

проблем створення еквівалентних мутантів. Мутанти вищого порядку створюються в результаті застосування декількох або більше мутацій для кожної мутованої версії програми [15]. Такий вид мутантів дозволяє позбутися зокрема еквівалентних мутантів і знизити в рази час виконання мутаційного тестування.

Проблема еквівалентного мутанта виникає, коли мутант, незважаючи на те, що він відрізняється від оригінального коду, веде себе ідентично до оригінальної програми. Іншими словами, мутант вводить зміну в код, яка не впливає на виведення чи поведінку програми, яку спостерігає тестовий набір. Оскільки мутант функціонально еквівалентний оригінальному коду, тестовий набір не може визначити його як дефектний варіант, що призводить до неправильного негативного результату.

Проблема еквівалентного мутанта становить значущий виклик, оскільки вона може знизити ефективність мутаційного тестування. Якщо набір тестів не може відрізнити еквівалентних мутантів від реальних дефектів, це може створити хибне уявлення про якість процесу тестування. Дослідники та практики в області тестування програмного забезпечення прагнуть розробити стратегії для вирішення проблеми еквівалентного мутанта та покращення надійності результатів мутаційного тестування.

Використання мутантів вищих порядків допомагає знизити кількість еквівалентних мутантів на 80 – 90% [16]. У дослідженні [17] прогнозування результатів мутаційного тестування здійснюється на основі мутацій другого порядку. З даних експериментів можна зробити висновок, що запропонований підхід може бути успішно застосований до прогнозування для повністю відмінного програмного забезпечення у порівнянні з програмним забезпеченням, на якому дана модель натренована.

Інший вид мутантів вищого порядку – супермутанти [18], що використовуються для зниження обчислювальних вимог, зокрема для зменшення витрат на тестовому наборі, що був машинно згенерований. Основною ідеєю такого підходу є поєднання мутантів, які взаємодіятимуть з малою вірогідністю. Супермутанти є одним із напрямків досліджень, що пропонується, при описі проблеми мутаційного тестування до фаззінгу [6]. При оцінці фаззерів мутаційним тестуванням використання цих мутантів показало прискорення в середньому у 3.8 разів.

Висновки

Незважаючи на те, що дані методи є досить новим напрямком дослідження, вони показують досить багатообіцяючі результати. Однак для покращення ефективності необхідно дослідити ряд факторів, що впливають на методи прогнозування результатів мутаційного тестування, а саме: ознаки, моделі машинного навчання, перевірка методів у різних середовищах, зокрема, імплементація методів на різних мовах, використання більш точних математичних моделей [7]. Як показує дослідження [8, 19] актуальним є пошук ознак, які мають більшу значущість, пошук нових метрик, а також використання різних методів для відбору ознак, а також для їх оцінки [9], пошук і використання ознак різного рівня масштабування - наприклад, ознак і метрик пакетного рівня, і ті що оцінюють якість коду на рівні програмних методів. Також відкритим залишається питання, чому певні метрики складності є більш прогнозовані, ніж інші та чому однакові метрики дають різний рівень прогнозованості в залежності від імплементації [12]. Для покращення швидкодії є сенс знаходження мінімального набору ознак [10] або використання, покращення і розширення легковісних ознак [11].

Список літератури

1. Laybourn I. M2: using mutation analysis to guide mutation-based fuzzing / I. Laybourn. — New York, NY, USA : Association for Computing Machinery, 2022.

2. Vikram V. Guiding greybox fuzzing with mutation testing / V. Vikram, I. Laybourn, A. Li, [et al.]. — Seattle WA USA : ACM, 2023.

3. Qian R. Investigating coverage guided fuzzing with mutation testing / R. Qian, Q. Zhang, C. Fang, L. Guo. — arXiv, 2022.

4. Görz P. Systematic assessment of fuzzers using mutation analysis / P. Görz, B. Mathis, K. Hassler, [et al.] // .

5. Just R. Are mutants a valid substitute for real faults in software testing? / R. Just, D. Jalali, L. Inozemtseva, [et al.]. — New York, NY, USA : Association for Computing Machinery, 2014.

6. Gopinath R. Mutation analysis: answering the fuzzing challenge / R. Gopinath, P. Görz, A. Groce. — arXiv, 2022.

7. Zhang P. CBUA: a probabilistic, predictive, and practical approach for evaluating test suite effectiveness / P. Zhang, Y. Li, W. Ma, [et al.] // IEEE Transactions on Software Engineering. — 2020. — P. 1–1.

8. Zhang J. Predictive mutation testing / J. Zhang, L. Zhang, M. Harman, [et al.] // IEEE Transactions on Software Engineering. — 2019. — Vol. 45, No. 9. — P. 898–918.

9. Aghamohammadi A. The threat to the validity of predictive mutation testing: the impact of uncovered mutants / A. Aghamohammadi, S.-H. Mirian-Hosseinabadi. — arXiv, 2020.

10. Duque-Torres A. Predicting survived and killed mutants / A. Duque-Torres, N. Doliashvili, D. Pfahl, R. Ramler. — Porto, Portugal : IEEE, 2020.

11. Grano G. Lightweight assessment of test-case effectiveness using source-code-quality indicators / G. Grano, F. Palomba, H. C. Gall // IEEE Transactions on Software Engineering. — 2021. — Vol. 47, No. 4. — P. 758–774.

12. Gil J. Better prediction of mutation score / J. Gil. — 2021.

13. Jalbert K. Predicting mutation score using source code and test suite metrics / K. Jalbert, J. S. Bradbury. — Zurich, Switzerland : IEEE, 2012.

14. Tan L. A model for predicting statement mutation scores / L. Tan, Y. Gong, Y. Wang // Mathematics. — 2019. — Vol. 7, No. 9. — P. 778.

15. Jia Y. Higher order mutation testing / Y. Jia, M. Harman // Information and Software Technology. — 2009. — Vol. 51, No. 10. — P. 1379–1393.

16. Papadakis M. An empirical evaluation of the first and second order mutation testing strategies / M. Papadakis, N. Malevris. — 2010.

17. Do V.-N. Predicting higher order mutation score based on machine learning / V.-N. Do, Q.-V. Nguyen, T.-B. Nguyen // Journal of Information and Telecommunication. — 2023. — P. 1–14.

18. Gopinath R. If you can't kill a supermutant, you have a problem / R. Gopinath, B. Mathis, A. Zeller. — Vasteras : IEEE, 2018.

19. Mao D. An extensive study on cross-project predictive mutation testing / D. Mao, L. Chen, L. Zhang. — Xi'an, China : IEEE, 2019.

A. Samoylenko

METHODS OF PREDICTING MUTATION TESTING RESULTS

Mutation testing is a widely recognized technique for assessing the effectiveness of test suites in detecting faults in software systems. Despite the effectiveness of this method, performing mutations requires significant computational expenses, which hinders its widespread adoption. In order to tackle the issue of efficiency, researchers have suggested different approaches such as selective, weakened, and predictive mutation testing. While various optimizations are utilized to cut costs, the current techniques and methods become unfeasible when dealing with an evaluation set that is not a static collection of tests but instead constantly evolving. This challenge arises due to the utilization of mutation testing as a metric to gauge the quality of a test suite developed through fuzzing. Both mutation testing and fuzzing are computationally intensive processes, creating a significant hurdle requiring innovative solutions. In light of these complexities, the optimization of computational resources becomes a key area of focus, aiming to balance efficiency without compromising the accuracy and depth of the testing methodologies employed. This intricate interplay between balancing costs, computational power, and testing effectiveness underscores the need for strategic optimization methods tailored to dynamic and resource-demanding evaluation scenarios. Finding approaches that allow for a quick and accurate assessment of mutation testing efficiency remains an open problem because mutation reduction techniques do not outweigh the random sampling method of mutants.

One way to address this issue is by utilizing prediction methods that forecast outcomes without executing mutations. These prediction methods for mutation testing represent a new research direction in the field of mutation testing optimization methods. The discussed methods can be classified based on the type of machine learning (supervised and unsupervised learning), types of features (dynamic and static), level of prediction (at the mutant level and test suite level), and the type of mutations used (first-order mutations and higher-order mutations).

Keywords: mutation testing, machine learning, deep learning, model, optimization, framework, data analysis.