УДК 004.054
DOI https://doi.org/10.32782/2663-5941/2023.3.1/30

Мойсеєнко О.В.
Івано-Франківський національний технічний університет нафти і газу

Гарасимів Т.Г.
Івано-Франківський національний технічний університет нафти і газу

ВИКОРИСТАННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ ПРИ ОПТИМІЗАЦІЇ ПРОЦЕСУ ФОРМУВАННЯ ТЕСТ-ПЛАНІВ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ

У роботі проаналізовано основні підходи, методи та алгоритми для автоматизованого створення тест-планів. Визначені основні переваги та недоліки. Сформовані основні вимоги до алгоритму автоматизованої генерації тестових випадків складних комп'ютерних систем на основі діаграм діяльності. Діаграма діяльності (AD) уніфікованої мови моделювання (UML) використовується для абстрактного представлення поведінки системи та використовується тестувальниками для створення тестових випадків і тестових даних. Під час проектування тестових випадків систем з одночасними (паралельними) діями діаграма активності може призвести до великої кількості шляхів, і не завжди можливо перевірити всі шляхи виконання. Зазвичай дослідження щодо отримання тестових випадків для систем із паралельними діями за допомогою AD зосереджено на звичайних методах пошуку, таких як пошук у ширину та пошук у глибину, виявляються неефективними. Щоб подолати цей недолік, в роботі пропонується метод із використанням парного тестування та генетичного алгоритму для отримання зменшеної кількості тестових випадків для комп'ютерних систем з одночасними діями. Запропоновано математичну модель, та отримано, як результат, алгоритм, за допомогою якого діаграма діяльності перетворюється у програму для автоматизованого генерування тестових випадків. Запропонований підхід та його реалізація може бути використаний для перевірки узгодженості між трасами виконання програми і поведінкою діаграм активності UML.

Ефективність модифікації алгоритму визначалась статистичним методами шляхом порівняння кількості тестів, досягнутих іншими методами, такими як пошук в ширину (BFS), пошук в глибину (DFS), повний, простий, випадковий шляхи. Модифікований підхід дає статистично обґрунтовані кращі результати з одночасним перехідним покриттям і попарним скороченням набору тестів. Експериментальні результати, отримані для п'яти різних AD, показують, що завдяки запропонованому підходу досягається 100% покриття одночасних переходів і більше ніж на 50% скорочення попарного набору тестів.

Ключові слова: *тестувальні випадки, діаграма діяльності, генетичний алгоритм, алгоритм попарного тестування.*

Постановка проблеми. Підвищення надійності програмного продукту є надзвичайно важливою та актуальною науковою задачею. Основним засобом вирішення даної проблеми є тестування програмного забезпечення (ПЗ), що вимагає значних затрат ресурсів, адже у більшості випадків проводиться вручну та повинне охоплювати усі сценарії використання і потоки управління та даних програмного продукту. Однак, зазвичай, це ручне тестування, що є трудомістким процесом без ефективної автоматизації, потребує багато часу та фінансових ресурсів, і не здатне забезпечити виявлення усіх відмов ПЗ. Отже, важливим завданням у сфері комп'ютерної інженерії є автоматизація процесу тестування, а саме побудова ефективних сценаріїв тестування, що дозволить скоротити часові та економічні затратити. Також важливо розробляти нові методи автоматизованої побудови сценаріїв тестування, що значно підвищить ефективність процесу тестування.

Аналіз останніх досліджень і публікацій. Однією з найновіших технологій для автоматизованої побудови сценаріїв тестування програмного забезпечення є тестування на основі моделі програмного продукту [1–3] – тестування програмного забезпечення (ПЗ), в якому тестові сценарії частково або повністю будуються з моделі, яка описує деякі аспекти (частіше функціональні) тестованої системи. Моделі програмної системи можуть відображати її поведінку або використовуватися для створення тестових стратегій чи середовища тестування.

Існує ряд моделей програмного забезпечення, що використовуються засобами автоматизованої побудови тестових сценаріїв. У загальному вони поділяються на чотири великі групи, які у свою чергу під час моделювання відповідно використовують: скінченні автомати, граф станів, UML діаграми та Марковські ланцюги. Окрім того існують моделі на основі дерева прийняття рішень, таблиць рішень та граматик, але вони є новими і не достатньо дослідженими [4].

Граф станів, так званий «потік тестування» використовується для моделювання програмних систем, побудови тестових сценаріїв та відображає реальне використання таких систем і, таким чином, є більш точним, ніж використання скінченних автоматів [5, 6]. Перевагами моделей даного класу є можливість їхнього застосування для багатопотокових програм, але все-таки такі моделі тяжко спроектувати та реалізувати для складних програмних систем.

Широкого застосування набуло тестування програмної системи на основі UML діаграм [7–10].

Зазвичай автоматизоване генерування тестових сценаріїв відбувається на основі діаграми станів [8], прецедентів [9], послідовності [10]. Основними перевагами таких підходів є можливість їх використання для складних систем, а також можливість моделювання паралельного програмного забезпечення.

Моделі на основі скінченних автоматів є відомими динамічними моделями для автоматизованої побудови тестів [11, 12]. Основними недоліками використання даних моделей є те, що їх легко спроектувати й підтримувати лише для систем із малим рівнем складності, неможливість використання даного класу моделей для паралельних ПЗ та систем з нескінченною кількістю станів.

Ще однією великою групою моделей для автоматизованої побудови тестових сценаріїв є моделі на основі дискретного Марковського ланцюга [13, 14]. За структурою Марковські ланцюги подібні до скінченних автоматів, але очевидною перевагою використання такого підходу є не лише генерація тестів, але й аналіз відмов для оцінки та прогнозування надійності програмного продукту. Сценарії тестування отримуються через послідовні переходи між компонентами у відповідності до ймовірностей передачі контролю. Генерування тестових сценаріїв може бути автоматизоване за допомогою хорошого генератора випадкових чисел на будь-якій мові програмування високого рівня.

Метою статті є удосконалення алгоритму генерації тестових випадків для зменшення тест-плану в процесі автоматизованого тестування складних комп'ютерних систем.

Виклад основного матеріалу. У випадку послідовного потоку керування тестові випадки генеруються шляхом трасування базових шляхів. Однак у випадку паралельних потоків відбувається розгалуження шляху через незалежний порядок виконання потоків. Існуючі методи в літературі [16, 20, 9] зменшують кількість одночасних тестів, використовуючи пошук спочатку в ширину (BFS), потім в глибину (DFS) і методи покриття шляху. Але ці методи не враховують випадкову взаємодію паралельних переходів.

Щоб досягти «перехідного» покриття за допомогою мінімального набору інтерактивних тестів, використаємо попарне тестування та генетичний алгоритм (GA). Попарне тестування [10–13] зменшує кількість тестів за допомогою статистичних методів. У запропонованому підході взаємодія одночасних дій розглядається для створення тестових випадків. На основі одночасних дій створюється набір парних тестів. GA використовується для подальшого зменшення кількості тестів у наборі парних тестів. GA є стохастичним пошуком і технікою оптимізації, яка використовується для проблем, пов'язаних із великим простором пошуку та складністю [19, 29].

В якості вхідних даних використаємо діаграму активності (AD), а вихідними даними буде згенеровано тестові випадки паралелізму. Одночасна система представлена такими компонентами AD, як вузли розгалуження, вузли приєднання та одночасні дії. Завдання полягає в тому, щоб створити тестові випадки, щоб перевірити природу паралелізму SUT. Метою запропонованого методу є знайти найкращий набір тестів із попарного набору тестів і досягти покриття всіх одночасних переходів. Тестові випадки будуть успішними, якщо набір тестів охоплює випадковість дій, які відбуваються, взаємодію одночасних переходів і всі одночасні дії та їхні переходи.

Пропонований підхід передбачає наступні кроки:

Генерація базового шляху: граф потоку керування (CFG) генерується з AD, а потім базові шляхи створюються за допомогою алгоритму DFS.

Попарна генерація тестів: переходи з різних гілок комбінуються для створення попарних тестів.

Оптимізація GA: попарний тестовий набір додатково скорочується за допомогою GA. Метою оптимізації є досягнення перехідного покриття.

Оптимізована генерація тестів: базові шляхи поєднуються зі скороченими тестами GA для створення оптимізованих наборів тестів.

AD аналізується за допомогою простого аналізатора API для XML (SAX), а потім ідентифікуються всі вузли та всі ребра. Відношення пріоритету між вузлами визначається шляхом сортування вузлів від початкового вузла до кінцевого вузла за допомогою алгоритму BFS.

Даний алгоритм додає суміжні вузли в чергу, а потім вилучає з черги кожен вузол, позначає як відвіданий і неодноразово перевіряє інші суміжні вузли. Черга очікування потрібна для збереження зв'язку пріоритету, коли різні шляхи зустрічаються в спільному вузлі, наприклад, вузол приєднання. Після сортування всіх вузлів, що стосуються зв'язку пріоритету, генерується CFG.

Базові шляхи отримують шляхом обходу CFG від початкового вузла до кінця за допомогою алгоритму DFS. Цей алгоритм є рекурсивним, він позначає всі пройдені вузли як відвідані. Коли зустрічається ребро з відвіданим вузлом, починається нова рекурсія, щоб отримати новий базовий шлях. Алгоритм завершується, коли пройдено всі ребра. Базові шляхи гарантують, що всі цикли проходять лише один раз, і всі вузли та переходи охоплені.

За допомогою генетичного алгоритму визначаються тестові випадки покриття переходів. Цей алгоритм зчитує «гени» з таблиці попарного введення. Він генерує випадкову популяцію, а потім обчислює придатність кожного, що допомагає визначити найкращих осіб. З найкращими особинами він виконує операції кросинговеру та мутації, щоб покращити фізичну форму під час наступного покоління. Цей алгоритм припиняє роботу, коли досягнуто максимальної кількості поколінь або знайдено найкращу особину.

Початкова популяція складається з 30 випадково згенерованих особин. Для наступного покоління розглядається близько 15 репродукцій. Після кожного покоління особини сортуються на основі значення придатності, розрахованого для кожного. «0-й» індивід завжди має найвищу придатність, а «1-й» індивід має наступне найвище значення і так далі. Для отримання кращої особини проводимо кросинговер і мутацію.

Функція пристосованості (fitness function) обчилюється наступним чином.

Fitness = fitness + 1 (якщо кожна пара відрізняється від наступної наступної пари).

Fitness = fitness + 1 (якщо пара тестових випадків знайдено в таблиці всіх пар тестових прикладів).

Fitness = fitness-кількість повторюваних пар (якщо та сама пара повторюється, накладаємо штраф).

Fitness = fitness/2 (якщо в ідентифікованих генах відсутній один перехід, накладається штраф).

Використовуємо кілька випадкових кросоверів (рекомбінацій). Точки перетину обчислюються на основі максимальної кількості переходів у будь-якій паралельній гілці.

Мутація. Якщо той самий ген повторюється в іншому тестовому прикладі, тоді тестовий випадок із дубльованим геном замінюється на тестовий випадок відсутнього гена, взятий із таблиці тестових випадків «Усі пари». Значення мутації вибрано нами як 0,01.

Для тестування алгоритму було використано готову розробку віртуальний інженер тестування (VTE) для аналізу AD і генерації тестів. Цей інструмент заснований на мові Java. Синтаксичний аналізатор SAX використовується для аналізу XMI-представлення AD, а потім генерується CFG. Для відображення графіка використовується інструмент GraphViz [18]. Усі вузли впорядковуються на основі їхнього пріоритету за допомогою алгоритму сортування на основі BFS. Інструмент «Усі пари» [15] використовується для створення комбінацій тестових наборів з одночасних вузлів і гілок.

Інструмент PWiseGen [25] модифіковано та використовується для зменшення кількості комбінацій набору тестів за допомогою техніки GA.

Набір метрик, створених VTE, далі аналізується та використовується для емпіричного дослідження.

Запропонований підхід було емпірично оцінено для визначення ефективності тесту щодо розміру набору тестів, покриття та продуктивності. У літературі доступні різні методи покриття, такі як покриття BFS [5, 9], покриття DFS [19], повне покриття шляху [3, 20, 21], просте покриття шляху [8, 17], базове покриття шляху [6], випадкове покриття [22] та одночасний пошук у черзі [23], які порівнюються з результатами нашого алгоритму. Ефективність алгоритму визначалась відносно часу, необхідного для пошуку розв'язку в секундах.

У запропонованому підході набір тестів було згенеровано за допомогою методу базового шляху. Паралельні набори тестів генеруються за допомогою інструменту. Ці попарні набори тестів зменшуються за допомогою техніки GA з одночасним покриттям переходів. Щоб отримати

скорочені тестові набори встановлюємо конфігурацію в інструменті PWiseGen згідно таблиці 1.

Було досліджено методи на основі оцінки BFS для створення оптимізованого набору тестів із оцінкою вузлів. Однак ці методи не враховують одночасну оцінку переходів і випадкове впорядкування. Ці набори тестів не фіксують дефекти, пов'язані з помилкою інтерфейсу серед дій у паралельній гілці. Ця характеристика методу BFS на основі оцінок представлена в таблиці 2.

Таблиця 1

Конфігурація GA

Параметри GA	Налаштування
кількість поколінь	200
чисельність популяцій	30
кількість репродукцій	15
значення мутації	0,1
кросоверний тип	множинний випадковий

Отже, методи на основі покриття DFS створюють оптимізований набір тестів із покриттям вузлів і переходів. Однак ці методи не враховують взаємодію одночасних дій і випадкове впорядкування. Ці набори тестів не фіксують дефекти, пов'язані з паралелізмами, такі як обробка ресурсів, обробка семафора та блокування/розблокування. Метод повного покриття шляху не генерує оптимізований набір тестів і не обробляє паралелізм. Прості методи на основі покриття шляхів генерують оптимізований набір тестів. Однак критерії тестового покриття, такі як охоплення активності та охоплення переходу, не гарантуються. Паралелізм також не обробляється. Метод на основі випадкового покриття є дорогим рішенням. Він базується на підході проб і помилок. Для досягнення більшого охоплення генерується більше випадкових сценаріїв. CQS корисний для оптимізації, обробка випадкового порядку та покриття вузлів. Покриття одночасного переходу не враховується.

Для проведення експериментів було обрано 14 рекламних оголошень, пов'язаних із транзакціями в банкоматах, онлайн-транзакціями, банківськими транзакціями тощо. Усі вони здійснювали одночасну діяльність. Для них побудовані відповідні діаграми, які й були використані в якості вхідних даних.

Для визначення ефективності модифікації алгоритму було обчислено всі можливі шляхи, позначені метрикою «Кількість можливих одночасних шляхів» для кожного AD. Інструмент VTE використовується для аналізу кожної реклами та створення тестових випадків базового шляху. З одночасних гілок і вузлів тестові набори «Усі пари» генеруються за допомогою інструмента «Усі пари». Зменшені тестові набори генеруються з попарних тестових наборів за допомогою нашого інструменту PWiseGen. Нарешті, тестові випадки базового шляху та скорочені тестові набори об'єднуються, щоб сформувати оптимізовані тестові набори. Базовий шлях, попарні, скорочені та оптимізовані тестові випадки позначаються параметрами, відомими як кількість тестових випадків базового шляху, кількість парних шляхів, кількість скорочених шляхів за GA та оптимізовані тестові випадки.

Можна побачити, що якщо кількість одночасних гілок і одночасних вузлів для кожної гілки зростає, то відповідні можливі одночасні шляхи також збільшуються. Якщо кількість одночасних вузлів у гілці становить 1 або 2, тоді можливих одночасних шляхів дуже мало. Якщо кількість одночасних вузлів у гілці більше 2, а кількість гілок більше 2, тоді можливих одночасних шляхів дуже багато. Зауважимо, що більшість AD, взятих з літератури, мають 1 або 2 одночасних вузла в гілці.

Значення вибуху траєкторії та оптимізовані тестові випадки та їхні відповідні середні значення наведені в табл. 3. Значення тестів на вибух нестабільні, тоді як значення оптимізованих тестів майже стабільні. Середні значення показують, що кількість тестів суттєво зменшилася.

Таблиця 2

Порівняння різних методів

Методи	Одночасна (випадкова) обробка	Попарні набори тестів	Одночасне покриття переходів	Одночасне покриття вузлів	Оптимізовані набори тестів
покриття BFS	X	X	X	√	√
покриття DFS	X	X	√	√	√
повне покриття шляху	X	X	√	√	X
просте покриття шляху	X	X	√	X	√
випадкове покриття	X	X	X	X	X
одночасний пошук у черзі	√	X	X	√	√
запропонований підхід	√	√	√	√	√

Таблиця 3

Порівняння кількості тестових випадків для різних алгоритмів

Приклад	Тестові випадки базових шляхів	Кількість тестових випадків оптимізованим алгоритмом
Мотиваційний приклад 1	34650	5
AD реєстрація пацієнта	6435	8
AD онлайн замовлення	2002	10
Мотиваційний приклад 2	15	11
AD банківська трансакція	20	2
Мотиваційний приклад 3	10	2
AD онлайн замовлення	4	3
AD покупки онлайн	3	3
AD ATM	10	3
AD ATM2	3	3
AD Facebook login	8	4
AD Online stock Exchange system	108	22
Середнє значення	3605,67	6,33

Кількість тестових випадків, обчислених для всіх пар і кількість оптимізованих тестів ми проаналізували за допомогою R Studio. У підсумку, можна зробити статистично обґрунтований висновок, що кількість тестових випадків згенерованих різними алгоритмами (один з яких нами модифікаований алгоритм) мають різні середні значення, наприклад 3606 і 6 відповідно. З ймовірністю >95% можна стверджувати, що середнє зменшення кількості тестових випадків є більше, ніж 600 разів.

Різні предметні програми можуть призвести до різних експериментальних результатів. Очікується, що запропонований підхід працюватиме з усіма видами одночасних структур. Експерименти проводились максимум із чотирма одночасними блоками. Час, потрібний GA для знаходження рішення, лінійно змінюється залежно від кількості блоків. Для вищого порядку кількості блоків поведінку GA потрібно вивчити. GA може зайняти більше часу, якщо кількість одночасних блоків перевищує десять. Можна зробити припущення, що розумна мутація, яка періодично змінює гени, вирішить цю проблему. Розумна мутація уникає локальних оптимумів і спрямовує GA до пошуку глобальних оптимумів. Тестові випадки потребують випадкового запуску одночасних дій. Механізм спрацьовування для реалізації цього тестового випадку може відрізнятися залежно від програми, що тестується. Ця проблема є з низьким ризиком, оскільки механізм запуску залежить від тригера введення, необхідного для запуску одночасної дії. Отже, визначення правильного тригера введення вирішить цю проблему.

Висновки. Розроблено математичну модель, та отримано як результат алгоритм, за допомогою якої діаграма діяльності перетворюється у програму для автоматизованого генерування тестових випадків. Запропонований підхід та його реалізація може бути використана для перевірки узгодженості між трасами виконання програми і поведінкою діаграм активності UML.

Удосконалено алгоритм генерації тестових випадків, шляхом використання попарного тестування та генетичного алгоритму, що дало можливість зменшити кількість тестових випадків, тоді як одночасні дії в складній системі призводять до вибуху шляху при використанні інших алгоритмів автоматизованого тестування. Запропонований підхід аналізує AD і генерує тестові випадки базового шляху. Попарні тестові набори генеруються з одночасних переходів різних паралельних гілок. Ці попарні набори тестів ще більше скорочуються з перехідним покриттям GA. Ці скорочені набори тестів поєднуються з тестовими випадками базового шляху з метою отримання оптимізованих тестових випадків. Ефективність модифікації алгоритму визначалась статистичним методами шляхом порівняння кількості тестів, досягнутих іншими методами, такими як BFS, DFS, повний шлях, простий шлях, випадковий і CQS. Модифікований підхід дає статистично обґрунтовані кращі результати з одночасним перехідним покриттям і попарним скороченням набору тестів. Експериментальні результати, отримані для п'яти різних AD, показують, що завдяки запропонованому підходу досягається 100% покриття одночасних переходів і більше ніж на 50% скорочення попарного набору тестів.

Результати експериментів показують, що час, необхідний для отримання розв'язку, прямо пропорційний кількості одночасних блоків в AD. Він включає час, витрачений на GA, і час попарного обчислення. Даний підхід займає 3–4 с у випадку одного одночасного блоку на AD. Він збільшується до 14 с, коли в AD присутні чотири одночасних блоки.

Досліджено п'ять AD з високим значенням паралельних шляхів і отримано попарне скорочення тестових випадків до 86%. Що стосується всіх можливих паралельних шляхів, спостерігалось скорочення оптимізованих тестових наборів на 99,99%.

Запропонований алгоритм досліджено в якості методу створення виконуваних тестових випадків з AD для функціональної перевірки системи. Даний метод дозволяє генерувати тестові сценарії з AD, які можуть безпосередньо виконуватися на цільовому пристрої.

Список літератури:

1. Broy M. Model Based Testing of Reactive Systems. / M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, A. Pretschner. LNCS 3472, Springer. 2005. 659 p.

2. Blackburn M. Why Model-Based Test Automation is Different and What You Should Know to Get Started./ M. Blackburn, R. Busser, A. Nauman. *In International Conference on Practical Software Quality*. 2004. P. 87-90.

3. Legeard B. Controlling Test Case Explosion in Test Generation from B Formal Models. / B. Legeard, F. Peureux, M. Utting. *The Journal of Software Testing, Verification and Reliability*. 2004. no. 14(2). Pp. 81–103.

4. Ebrahim Shamsoddin-Motlagh. A Review of Automatic Test Cases Generation. *International Journal of Computer Applications*. 2012. no. 57(13). Pp. 25-29.

5. Belli F. A Holistic Approach to Testing of Interactive Systems using Statecharts. / F. Belli, C.J. Budnik, A. Hollman. *Proceedings of 2nd South-East European Workshop on Formal Methods (SEEFM 05),* South-Eastern European Research Center SEERC. 2005. Pp. 1–15.

6. Hyoung Seok Hong. Automatic Test Generation from Statecharts Using Model Checking / Insup Lee, Oleg Sokolsky, Sung Deok Cha. *In Proceedings of FATES'01, Workshop on Formal Approaches to Testing of Software*. – 2001.

7. Hu Y.T. Automatic Black-Box Method-Level Test Case Generation Based on Constraint Logic Programming. / Y.T. Hu, N.W. Lin // *Computer Symposium (ICS)*. 2010. Pp. 977-982.

8. Samuel P. (2008). Automatic test case generation using unified modeling language (UML) state diagrams. / P. Samuel, R. Mall, A. Bothra. *The Institution of Engineering and Technology, IET Softw.* 2008. no. 2. Pp. 79–93.

9. Sarma M. Automatic Test Case Generation from UML Models. / M. Sarma, R. Mall. *10th International Conference on Information Technology*. 2007. Pp. 196-201.

10. Sarma M. Automatic Test Case Generation from UML Sequence Diagrams. / M. Sarma, D. Kundu, R. Mall. *15th International Conference on Advanced Computing and Communications.* 2007. Pp.60-65.

11. Santiago V. An Environment for Automated Test Case Generation from Statechart-based and Finite State Machine-based Behavioral Models. / V. Santiago, N. Vijaykumar, D. Guimaraes. *IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08).* 2008. Pp.63-72.

12. Susumu Fujiwara. Test Selection Based on Finite State Models. / Susumu Fujiwara, Gregor Bochmann, Ferhat Khendek, Mokhtar Amalou, Abderrazak Ghedamsi. *IEEE Transactions on Software Engineering*. 1991. no. 6(17). Pp. 591–603.

13. Whittaker J. A Markov Chain Model for Statistical Software Testing. / J. A. Whittaker, M. G. Thomason. *Software Engineering, IEEE Transactions*. 1994. Pp. 812-824.

14. Winfried Dulz. Matelo - Statistical Usage Testing by Annotated Sequence Diagrams, Markov Chains and TTCN-3 / Winfried Dulz, Fenhua Zhen. *Third International Conference On Quality Software*. – 2003.

15. Felderer M.'Manual test case derivation from UML activity diagrams and state machines: a controlled experiment'/ Felderer, M., Herrmann, A., Inf. Softw. Technol., 2015, 61, pp. 1–15.

16. Anbunathan R. Automatic test generation from UML sequence diagrams for android mobiles/ Anbunathan R., Basu A, Int. J. Appl. Eng. Res., 2016, 11, (7), pp. 4961–4979.

17. Shirole, M. Generation of improved test cases from UML state diagram using genetic algorithm/ Shirole, M., Suthar, A., Kumar, R. Proc. Fourth India Software Engineering Conf., Thiruvananthapuram, India, 2011, pp. 125–134

18. Kundu, D. A novel approach to generate test cases from UML activity diagrams/ Kundu, D., Samanta, D. J. Object Technol., 2009, 8, (3), pp. 65–83

19. Linzhang, W. Generating test cases from UML activity diagram based on gray-box method/ Linzhang, W., Jiesong, Y., Xiaofeng, Y. IEEE Proc. 11th Asia-Pacific Software Engineering Conf. (APSEC), Busan, Korea, 2004, pp. 1–8

20. Mingsong, C. Automatic test case generation for UML activity diagrams/ Mingsong, C., Xiaokang, Q., Xuandong, L. 2006 Int. Workshop on Automation of Software Test, Shanghai, China, 2006, pp. 2–8

21. Shirole, M., Kommuri, M., Kumar, R.: 'Transition sequence exploration of UML activity diagram using evolutionary algorithm'. Proc. ISEC '12, Kanpur, India, 2012, pp. 22–25

22. Flores, P., Cheon, Y.: 'PWiseGen: generating test cases for pairwise testing using genetic algorithms', Departmental Technical Reports (CS) Paper 595, 2011

23. Nguyen, C.D., Marchetto, A., Tonella, P.: 'Combining model-based and combinatorial testing for effective test case generation'. Proc. 2012 Int. Symp. Software Testing and Analysis (ISSTA), Minneapolis, MN, USA, 2012.

Moyseenko O.V., Harasymiv T.G. USE OF GENETIC ALGORITHMS IN OPTIMIZING THE PROCESS OF FORMING COMPUTER SYSTEM SOFTWARE TEST PLANS

The article analyzes the main approaches, methods and algorithms for the automated creation of test plans. The main advantages and disadvantages are defined. The main requirements for the algorithm of automated generation of test cases of complex computer systems based on activity diagrams have been formulated. A Unified Modeling Language (UML) activity diagram (AD) is used to abstractly represent the behavior of a system and is used by testers to generate test cases and test data. When designing test cases for systems with simultaneous (parallel) actions, an activity diagram can lead to a large number of paths, and it is not always possible to test all execution paths. Usually, research on obtaining test cases for parallel systems using AD focuses on conventional search methods such as breadth-first search and depth-first search, which are found to be inefficient. To overcome this shortcoming, the paper proposes a method using paired testing and a genetic algorithm to obtain a reduced number of test cases for computer systems with simultaneous actions. The authors proposed a mathematical model, and as a result, an algorithm was obtained, with the help of which the activity diagram is transformed into a program for the automated generation of test cases. The proposed approach and its implementation can be used to check the consistency between program execution paths and the behavior of UML activity diagrams.

The effectiveness of the algorithm modification was determined by statistical methods by comparing the number of tests achieved by other methods such as BFS, DFS, full path, simple path, random and CQS. The modified approach gives statistically justified better results with simultaneous transient coverage and pairwise reduction of the test set. Experimental results obtained for five different ADs show that the proposed approach achieves 100% coverage of simultaneous transitions and more than 50% reduction in the pairwise test set.

Key words: *test cases, activity diagram, genetic algorithm, pairwise testing algorithm.*