МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 58.00.00.000 ПЗ

Група ІПм-23-3

Саган Валентин

2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Саган Валентин Михайлович

(прізвище, ім'я, по батькові)

УДК 004.942

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі, методи та засоби інтерактивного тестування програмного забезпечення

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Саган В.М.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Піх Володимир Ярославович, к.т.н., доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2024

Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ 04 ” вересня 2024 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Сагану Валентину Михайловичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі, методи та засоби інтерактивного тестування програмного забезпечення”

керівник проекту (роботи) Піх Володимир Ярославович, к.т.н., доцент

затверджені наказом закладу вищої освіти від “ 22 ” листопада 2024 р. № 781/7

2. Строк подання студентом проекту (роботи) 15 грудня 2024 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій тестування ПЗ

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Дослідження та теоретичний опис процесів тестування програмного забезпечення
2. Методи інтерактивного еволюційного пошуку під час тестування програмного забезпечення
3. Особливості побудови системи інтерактивного тестування програмного забезпечення
4. Техніка та методологія оцінки інтерактивного тестування програмного забезпечення

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Узагальнений еволюційний алгоритм (рис. 1.1)
2. Схема функціонування інтерактивної система тестування ПЗ на основі пошуку (рис. 1.2)
3 Огляд інтерактивної система тестування ПЗ на основі пошуку (рис. 1.3)
4. Інтерфейс інструменту EvoSuite (рис. 1.4)
5. Структура методики інтерактивного тестування (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2024 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2024	виконано
2	Аналіз концепцій та алгоритмів предметної області	29.09.2024	виконано
3	Дослідження та теоретичний опис процесів тестування програмного забезпечення	15.10.2024	виконано
4	Методи інтерактивного еволюційного пошуку під час тестування програмного забезпечення	08.11.2024	виконано
5	Особливості побудови системи інтерактивного тестування програмного забезпечення	20.11.2024	виконано
6	Техніка та методологія оцінки інтерактивного тестування програмного забезпечення	01.12.2024	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2024	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 77 с., 20 рис., 4 табл., 52 джерел.

Тема: Моделі, методи та засоби інтерактивного тестування програмного забезпечення

Об'єкт дослідження: процес тестування програмного забезпечення з використанням інтерактивних методів.

Мета роботи: розробка концепцій побудови інтерактивної системи тестування програмного забезпечення, заснованої на еволюційних методах пошуку, та оцінка її ефективності шляхом імітаційного експерименту.

Предмет дослідження: методи та алгоритми інтерактивного еволюційного пошуку, що застосовуються у тестуванні програмного забезпечення.

Результати дослідження

В роботі розроблено підхід до інтерактивного тестування програмного забезпечення, заснований на поєднанні метаевристичних методів пошуку та інтерактивних стратегій і запропоновано концепцію побудови інтерактивної системи тестування з внутрішнім і зовнішнім циклами роботи.

Висновок

Розроблено алгоритм пошуку, адаптований для використання в інтерактивному середовищі тестування і проведено емпіричну оцінку прототипу системи, яка підтвердила його ефективність.

ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ, АЛГОРИТМ ПОШУКУ, ІНТЕРАКТИВНИЙ ЕВОЛЮЦІЙНИЙ ПОШУК, МЕТАЕВРИСТИЧНІ МЕТОДИ, ІНТЕРАКТИВНА СИСТЕМА ТЕСТУВАННЯ, ЯКІСТЬ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ABSTRACT

Master Thesis: 77 pp., 20 fig., 4 tab., 52 sources.

Thesis Subject: Models, methods and tools of interactive software testing

Object of research: the process of software testing using interactive methods.

Purpose of work: development of concepts for building an interactive software testing system based on evolutionary search methods and evaluation of its effectiveness through a simulation experiment.

Subject of research: methods and algorithms of interactive evolutionary search used in software testing.

Research results

The paper develops an approach to interactive software testing based on a combination of metaheuristic search methods and interactive strategies and proposes a concept for building an interactive testing system with internal and external work cycles.

Conclusion

A search algorithm adapted for use in an interactive testing environment is developed and an empirical evaluation of the system prototype is carried out, which confirms its effectiveness.

SOFTWARE TESTING, SEARCH ALGORITHM, INTERACTIVE EVOLUTIONARY SEARCH, METAHEURISTIC METHODS, INTERACTIVE TESTING SYSTEM, SOFTWARE QUALITY

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SBST - Search-Based Software Testing

SBSE - Search-Based Software Engineering

ISBST - Interactive Search-Based Software Testing

IEC - Interactive Evolutionary Computation

TTM - Technology Transfer Model

BDD – Behavior-Driven Development

TDD – Test-Driven Development

SUT – System Under Test

UAT – User Acceptance Testing

RTM – Requirements Traceability Matrix

PIT – Process Improvement Testing

VST – Visual Software Testing

DUT – Device Under Test

MTTF – Mean Time to Failure

FTR – First Time Right

ISTQB – International Software Testing Qualifications Board

ATP – Automated Testing Practices

ALM – Application Lifecycle Management

CMMI – Capability Maturity Model Integration

E2E – End-to-End Testing

ATDD – Acceptance Test-Driven Development

CBT – Component-Based Testing

COTS – Commercial Off-The-Shelf

FT – Functional Testing

MMT – Model-Based Testing

OAT – Operational Acceptance Testing

RBT – Risk-Based Testing

ВСТУП

Актуальність теми.

З кожним роком масштаби та складність програмного забезпечення стрімко зростають, що підвищує вимоги до його якості, продуктивності та надійності. Помилки в програмному забезпеченні можуть спричинити значні фінансові втрати, порушення безпеки даних і зниження довіри користувачів. У зв’язку з цим тестування програмного забезпечення стає ключовим етапом у процесі його розробки.

Традиційні методи тестування часто виявляються недостатньо ефективними для роботи зі складними системами через високу ресурсоємність і обмежену адаптивність. У відповідь на ці виклики інтерактивні системи тестування, засновані на еволюційних підходах, пропонують інноваційні рішення. Вони поєднують можливості метаевристичних алгоритмів для оптимізації пошуку дефектів із інтерактивністю, що дозволяє враховувати специфіку конкретного проєкту або середовища.

З іншого боку, інтерактивне тестування вимагає більш глибокої теоретичної бази та розробки практичних інструментів. Існує необхідність створення таких систем, які б могли автоматично адаптувати стратегії пошуку до динамічних змін програмного середовища та вимог замовників. Особливо актуальними є дослідження, спрямовані на розробку універсальних рішень, які б могли використовуватись у різних доменах — від мобільних застосунків до складних корпоративних систем.

Таким чином, розвиток інтерактивного тестування на основі еволюційних методів є не лише актуальним, але й нагальним завданням, яке має стратегічне значення для програмної інженерії. Це дослідження спрямоване на розв'язання цих проблем шляхом створення концептуально нової інтерактивної системи тестування програмного забезпечення, здатної підвищити ефективність і якість процесу тестування.

Мета дослідження – розробка концепцій побудови інтерактивної системи тестування програмного забезпечення, заснованої на еволюційних методах пошуку, та оцінка її ефективності шляхом імітаційного експерименту.

Об'єкт дослідження - процес тестування програмного забезпечення з використанням інтерактивних методів.

Предмет дослідження - методи та алгоритми інтерактивного еволюційного пошуку, що застосовуються у тестуванні програмного забезпечення.

Задачі дослідження:

- Аналіз існуючих методів тестування програмного забезпечення, зокрема інтерактивних та еволюційних підходів.

- Формулювання основних концепцій побудови інтерактивної системи тестування.

- Розробка методики застосування інтерактивного еволюційного пошуку у тестуванні програмних продуктів.

- Створення прототипу інтерактивної системи тестування та емпірична оцінка її ефективності.

- Проведення імітаційного експерименту та аналіз отриманих результатів.

Методи дослідження

1. Методи аналізу та синтезу для формулювання концепцій і методології.

2. Еволюційні метаевристичні алгоритми для розробки інтерактивної системи.

3. Емпіричний метод для оцінки ефективності прототипу.

Наукова новизна отриманих результатів

Розроблено підхід до інтерактивного тестування програмного забезпечення, заснований на поєднанні метаевристичних методів пошуку та

інтерактивних стратегій і запропоновано концепцію побудови інтерактивної системи тестування з внутрішнім і зовнішнім циклами роботи.

Практичне значення результатів

Результати дослідження можуть бути використані в розробці програмного забезпечення для автоматизації тестування, що підвищує його якість і знижує витрати часу. Створена методологія може застосовуватись як у комерційних, так і в освітніх проєктах для оптимізації процесу тестування.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 77 сторінок, і містить 20 рисунків, 4 таблиці, список використаних джерел із 52 позицій.

РОЗДІЛ 1. ДОСЛІДЖЕННЯ ТА ТЕОРЕТИЧНИЙ ОПИС ПРОЦЕСІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1. Основні відомості про тестування програмного забезпечення на основі пошуку

Програмне забезпечення все більше присутнє в різноманітних системах і продуктах у різних сферах. Часто це набуває форми вбудованого програмного забезпечення, де саме програмне забезпечення є лише одним із компонентів продукту. Є приклади вбудованого програмного забезпечення, яке використовується в промислових сферах [1], від автомобільної техніки до аерокосмічної техніки, оборони тощо [2, 3].

Розробка та забезпечення якості вбудованого програмного забезпечення також потребує адаптації до домену. Обмеження доступного апаратного забезпечення впливатимуть на програмний компонент. Деякі продукти з інтенсивним використанням програмного забезпечення, наприклад аерокосмічні чи автомобільні програми, можуть мати дуже суворі стандарти забезпечення якості та вимагати спеціальних процесів для забезпечення відповідності. У результаті програмне забезпечення все частіше розробляється інженерами домену або системними інженерами; люди з різним освітнім середовищем і стилями [4], чия підготовка та досвід зосереджені більше на розумінні контексту та домену, а не на розробці та тестуванні програмного забезпечення. Далі ми будемо називати таких інженерів «спеціалістами в галузі», щоб підкреслити вирішальну роль їхніх знань і навичок, пов’язаних із предметною областю, і відносно другорядну природу їхньої ролі розробки програмного забезпечення.

Тестування програмного забезпечення є необхідною діяльністю, але часто вважається трудомісткою: часто вважається, що вона займає до половини вартості проекту [5]. Хоча точні зусилля є предметом дебатів, тестування програмного забезпечення є важливою частиною процесу

забезпечення якості та часто потребує ресурсів. У контексті, коли програмне забезпечення не є основним компонентом продукту, ресурси для тестування програмного забезпечення можуть стати дефіцитними. Оскільки багато вбудованих систем підпадають під жорсткі обмеження щодо вартості [2, 6], для компаній може стати непомірно дорогим адаптувати існуючі методи тестування програмного забезпечення, розробити власні методи тестування програмного забезпечення або навіть оцінити ступінь виконання тестування програмного забезпечення та якщо є потреба у вдосконаленні.

Забезпечення якості в цьому сценарії набуває форми тестування системи та приймального тестування, оцінки загальної системи наприкінці циклу розробки. Це розумно, оскільки якість системи залежить від якості всіх компонентів, а не лише програмного забезпечення. Тим не менш, пізнє виявлення помилок програмного забезпечення призводить до збільшення вартості, оскільки компоненти потрібно переробляти. Затримки та переробка через несправності, якими б незначними вони не були, ймовірно також поширюватимуться на інші компоненти, ще більше збільшуючи витрати та затримки. Більше того, вбудоване програмне забезпечення часто можна знайти в системах, які підпорядковуються правилам безпеки, які мають важливе значення для життя або місії. Таким чином, проблема полягає в забезпеченні високого рівня якості програмного забезпечення в складному контексті, який потребує знань як у домені, так і в тестуванні програмного забезпечення, при цьому гарантуючи, що вартість утримується на прийнятному рівні.

Інженерія програмного забезпечення на основі пошуку — це термін, який описує застосування метаевристичних методів до проблем інженерії програмного забезпечення [7]. Для окремого випадку тестування ці підходи описуються як тестування програмного забезпечення на основі пошуку (SBST - Search-Based Software Testing) [8, 9].

В ідеалі рішення для тестування програмного забезпечення для певної області поєднує передову практику тестування програмного забезпечення зі

знаннями домену та досвідом процедур і правил забезпечення якості для конкретної програми. SBST було запропоновано та підтверджено в академічних колах для низки різних застосувань. Є, однак, кілька прикладів систем SBST, які застосовуються в промисловості та передаються для промислового використання. У цій роботі пропонується рішення, яке має на меті надати фахівцям предметної області спосіб використання методів тестування програмного забезпечення, не вимагаючи додаткового навчання в іншій галузі. Для досягнення цього ефекту рішення, запропоноване в цій роботі, спирається на взаємодію між фахівцем предметної області та системою SBST на основі концепції інтерактивних еволюційних обчислень (IEC - Interactive Evolutionary Computation) [10]. Рішення складатиметься із застосування SBST в інтерактивній формі та перенесення отриманого інструменту в промисловий контекст. Це рішення називається системою інтерактивного тестування програмного забезпечення на основі пошуку (ISBST).

Точні деталі алгоритму SBST можуть відрізнятися, але рисунок 1.1 ілюструє основні ідеї SBST на прикладі популяційного генетичного алгоритму.

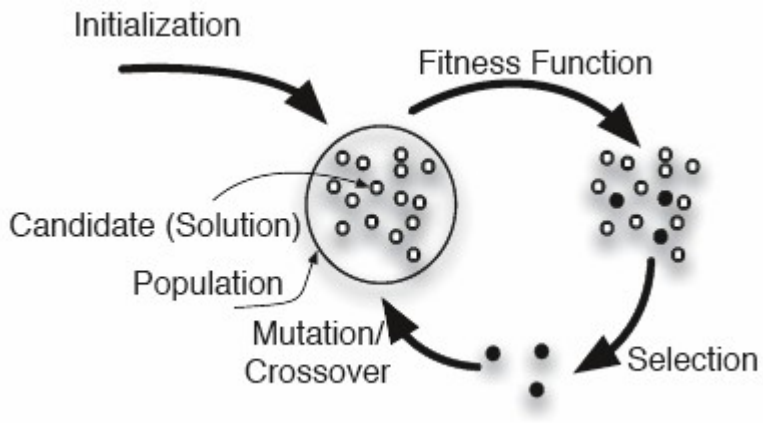


Рис. 1.1. Узагальнений еволюційний алгоритм

Алгоритм, показаний на рисунку 1.1, складається з кількох кроків.

1. Ініціалізація. Спочатку генерується початкова популяція варіантів рішень. Це часто робиться випадковим чином, але також можна застосувати більш складні методи.

2. Оцінка фізичної форми. Кожне рішення-кандидат у популяції оцінюється за допомогою функції відповідності. Функція Fitness призначає числове значення кожному кандидату рішення та дозволяє порівнювати комплексні кандидати.

3. Вибір. З початкової сукупності варіантів рішень вибирається підмножина для використання в наступному поколінні. Відбір надає перевагу рішенням-кандидатам із кращою придатністю, але інші кандидати також мають ймовірність бути обраними.

4. Генерація населення. Відібрані кандидати складають основу для створення нового покоління. Це робиться за допомогою генетичних - операторів. Два приклади таких генетичних операторів - мутація та кросинговер. Мутація передбачає випадкову модифікацію рішення-кандидата. Кросовер передбачав створення нових рішень-кандидатів шляхом комбінування існуючих.

Для SBST окремим рішенням-кандидатом може бути траса тестового запуску, тестовий приклад або набір тестових даних. Генетичний алгоритм вибере тестові випадки, які краще відповідають критеріям якості, закодованим у функції відповідності. Очікується, що протягом кількох поколінь загальна підготовленість популяції кандидатів покращиться.

1.2. Інтерактивна система тестування програмного забезпечення на основі пошуку

У цьому розділі коротко описано компоненти системи ISBST (Interactive Search-Based Software Testing) у її найновішій і найповнішій формі, а також їхнє відношення до SBSE загалом.

Інструмент ISBST — це інструмент тестування програмного забезпечення на основі пошуку, призначений для того, щоб спеціалісти домену могли використовувати свої знання та досвід для керування пошуком. Ці вказівки досягаються, дозволяючи фахівцю домену змінювати функцію відповідності, яка керує пошуком, а потім оцінювати отримані тестові приклади для подальшого вдосконалення свого визначення функції відповідності. Функція відповідності складається з ряду критеріїв, які називаються цілями пошуку, які вимірюють характеристики SUT. Спеціаліст домену керує пошуком, визначаючи відносну важливість цих цілей.

Система ISBST має два вкладених компоненти: система SBST, яка підключається до SUT , утворює внутрішній цикл, а зовнішній цикл забезпечує взаємодію між внутрішньою системою SBST і фахівцем домену. Огляд системи ISBST можна побачити на рисунку 1.2.

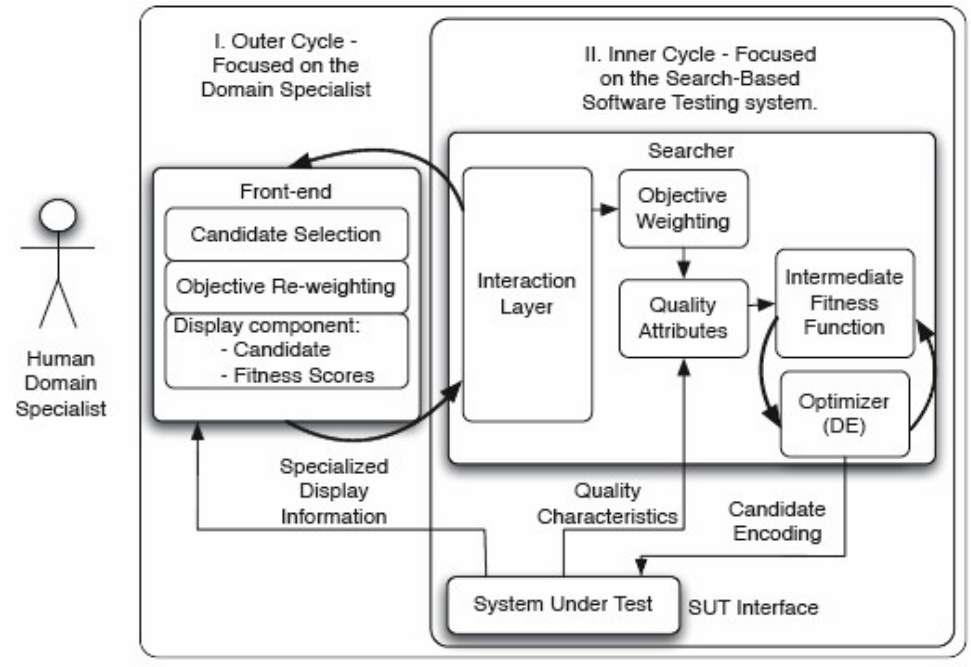


Рис. 1.2. Схема функціонування інтерактивної система тестування програмного забезпечення на основі пошуку

Внутрішній цикл складається з самого алгоритму пошуку, функції придатності та цілей пошуку, які нею керують, а також механізму, який

обробляє взаємодію з SUT. Використовується алгоритм диференціального еволюційного алгоритму, який генерує задану кількість вхідних даних тестового випадку, які потім використовуються для запуску SUT і отримання відповідної поведінки. Кожен вхід тесту складається з вектора дійсних чисел. Поєднання вхідних даних і поведінки разом називають кандидатом. Після реєстрації поведінки кандидата оцінюють за допомогою функції придатності.

Стратегія мутації для нових кандидатів така:

$$v_{i,G+1} = x_{r_1,G} + F \times (x_{r_2,G} - x_{r_3,G})$$

де r1, r2, r3 Є 1, 2,..., NP, є цілими числами, взаємно різними і відмінними від поточного індексу i. F – дійсний і постійний коефіцієнт Є (0, 2], який контролює посилення диференціальної варіації (x_{r2},G - x_{r3} , G). Якщо мутантний вектор є вдосконаленням цільового вектора, він замінює його в наступному поколінні.

Функція придатності складається з кількох цілей пошуку, які оцінюються незалежно. Результати кожної з цих оцінок збираються та комбінуються відповідно до суми зважених глобальних коефіцієнтів Бентлі [10], як показано нижче:

$$\mathrm{DAFF}_j = \sum_{i=1}^{\mathrm{nObjectives}} \mathrm{Weight}_i * \mathrm{Value}_{i,j}$$

де $DAFF_j$ - це значення придатності кандидата j, Weighti - це поточна вага цілі i, а $Value_{i,j}$ - це значення придатності кандидата j, виміряне за ціллю i. Значення DAFFj є сумою зважених значень придатності для всіх nObjectives цілей. Ціллю k можна відмовитися від обчислення, встановивши Weightk = 0.

Зовнішній цикл є оболонкою навколо системи SBST, яка дозволяє фахівцям предметної галузі взаємодіяти з SBST, коригуючи відносну важливість кожної цілі пошуку та переглядаючи отримані кандидати. Кандидати, отримані в результаті пошуку, відображаються як група, відносно значень придатності, які вони отримали. Кожного окремого кандидата можна відобразити детальніше, якщо фахівець предметної галузі вважає це корисним. Взаємодія пошуку здійснюється шляхом дозволу фахівцю предметної галузі встановлювати відносні ваги для кожної цілі пошуку.

Потім ваги передаються Внутрішньому циклу, де вони є частиною оцінки придатності.

На даний момент нові цілі пошуку можна додавати лише вручну, з кодом для оцінки придатності, доданим до відповідного модуля. Однак, як тільки код буде написаний, нові цілі пошуку автоматично використовуватимуться для майбутніх оцінок придатності. Однак досвід показав, що будь-який набір цілей пошуку, який попередньо визначений, навряд чи буде повним, тому засіб дозволу додавання нових цілей був би корисним для практичного розгортання та подальшої оцінки.

1.3. Основні концепції системи тестування на основі пошуку

Тестування програмного забезпечення на основі пошуку (SBST) — це застосування методів пошуку до проблеми тестування програмного забезпечення. SBST застосовувався до різноманітних проблем тестування [8, 9], від об’єктно-орієнтованих контейнерів [24] до мов динамічного програмування [25] та інваріантів [28]. SBST є частиною ширшої області розробки програмного забезпечення на основі пошуку (SBSE), терміну, введеному в [7]. Оскільки підходи, засновані на пошуку, застосовувалися протягом життєвого циклу розробки програмного забезпечення [26], розумно очікувати, що будь-які висновки, що стосуються загальної області SBSE, можна застосувати до окремого випадку SBST.

Контекст системи, дослідженої в цій роботі, полягає в тестуванні програмного забезпечення, що виконується фахівцями в галузі, які мають відносно невеликі знання про тестування програмного забезпечення або методи пошуку. Фахівці домену мали б широкі знання про можливості системи, що розробляється, її власний контекст і обмеження, накладені на неї, а також про фокуси якості, які вони мали б шукати для кожного компонента цієї системи.

Поєднання неспеціалістів-розробників програмного забезпечення та важливості знань предметної області та обмежень робить непрактичним розробку фітнес-функції заздалегідь. Щоб розробити відповідну функцію придатності для тестованого компонента, фахівець із домену мав би взаємодіяти з системою та вносити корективи у критерії, що використовуються.

Щоб краще проілюструвати концепції, які обговорюються в цьому розділі, ми наведемо анонімний промисловий приклад. Додаток, який ми будемо використовувати, — це програма керування, яка дозволяє керувати джойстиком або набором джойстиків механічною рукою. Вхідними для програмного забезпечення контролера є сигнали джойстика та датчики, які вказують швидкість кошика на кінці крана. Виходи - це сигнали для гідравлічних насосів, які приводять у рух важіль.

Тестована система (SUT) у цьому випадку є програмним забезпеченням для компонента контролера. Метою системи тестування програмного забезпечення на основі пошуку є створення тестових випадків, які гарантують відповідність системи стандартам якості, гарантують відсутність порушень обмежень і виявлення будь-яких додаткових помилок або неочікуваної поведінки.

Система тестування програмного забезпечення на основі пошуку є результатом застосування методології, представленої в цій роботі, у відповідній компанії. Система призначена для адаптації до конкретного контексту та компанії, у якій вона, як очікується, функціонуватиме, але є

достатньо загальною, щоб спеціалісти домену могли тестувати нові програми в межах цього контексту.

На рисунку 1.3 показано структуру комплексної системи тестування програмного забезпечення на основі інтерактивного пошуку.

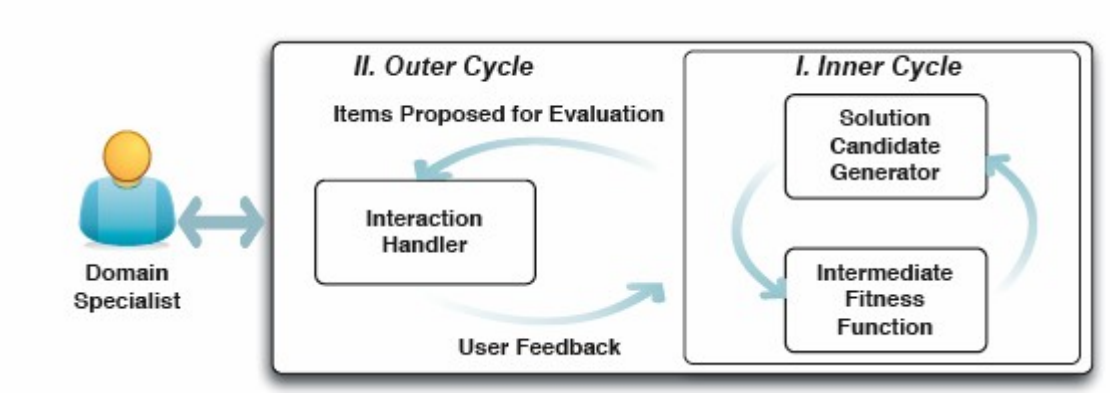


Рисунок 1.3. Огляд інтерактивної системи тестування програмного забезпечення на основі пошуку (ISBSE)

Зовнішній цикл є інтерактивною системою, заснованою на пошуку, яка використовує людину-фахівця предметної галузі як функцію придатності. Він посередничає у взаємодії між фахівцем предметної галузі та системою за допомогою компонента, який називається "Обробник взаємодії". Для цілей цього обговорення ми називаємо потенційним рішенням або кандидатом на рішення будь-яку особу, яка є частиною популяції, яку очікується оцінити людиною-користувачем.

Метою Обробника взаємодії є відображення потенційних рішень, показаних людині-фахівцю предметної галузі, та збір їхнього зворотного зв’язку. Зворотний зв’язок у типі пропонованої системи може стосуватися трьох окремих питань:

- Зворотний зв’язок щодо кандидата на рішення. Це описує зворотний зв’язок, пов’язаний із кандидатами на рішення. Окрім вибору потенційних рішень для наступного покоління, людина-фахівець предметної галузі може призначати значення кожному кандидату на рішення, який вони вибирають для наступного покоління, надаючи їм еволюційну перевагу.

- Зворотний зв’язок щодо відображення. Це описує зворотний зв’язок, пов’язаний зі способом відображення кандидатів на рішення, кількістю відображених кандидатів і будь-якою додатковою інформацією, яка доступна або може бути надана. Беручи до уваги наш поточний приклад, обговорений вище, фахівець предметної галузі може вирішити побачити необхідну пам’ять, час відповіді для вихідних сигналів або розбіжності між очікуваними вихідними сигналами та фактичними вихідними сигналами, на додаток до статусу проходження або невдачі кожного набору.

- Зворотний зв’язок щодо фокусу якості. Оскільки система, заснована на пошуку, може генерувати більше кандидатів на рішення, ніж очікується оцінити людиною, існує деякий внутрішній механізм, який дозволяє попередній вибір потенційних рішень. Цей тип зворотного зв’язку дозволяє фахівцю предметної галузі встановлювати або змінювати критерії, за якими здійснюється цей попередній вибір. У міру просування пошуку відповідних тестових випадків може виникнути необхідність скоригувати фокуси якості, які встановлюють критерії відбору. Наприклад, початковою вимогою до джойстика в нашому прикладі може бути відповідний час або точність вихідного сигналу. Після того, як модуль буде визнано задовільним з цієї точки зору, пошук великих відхилень або небажаної поведінки може стати більш важливим. Цей тип зворотного зв’язку дозволив би фахівцю предметної галузі змінити фокус пошуку без перезапуску пошуку, тим самим зберігши характеристики кандидатів на рішення, які вже є в популяції.

Заміна функції придатності людиною-фахівцем предметної галузі обмежує кількість потенційних рішень, які система може обробити таким чином. Додаткова інформація, яку може надати людина, є спробою компенсувати меншу кількість кандидатів на рішення, що обробляються, шляхом покращення механізмів вибору всередині системи.

Внутрішній цикл є системою тестування програмного забезпечення, заснованою на пошуку, яка використовує гнучку функцію придатності. Метою цієї системи є генерування та вибір найкращих кандидатів на рішення

для оцінки людиною-фахівцем предметної галузі. Такий підхід дозволяє системі досліджувати ширший простір рішень, одночасно дозволяючи людині-фахівцю предметної галузі застосовувати свій досвід і розуміння.

Сам внутрішній цикл має два компоненти:

Компонент пошуку. Метою компонента пошуку є інкапсуляція алгоритму, який створює нове покоління потенційних рішень. Інкапсуляція цього компонента дозволяє змінити існуючий алгоритм, якщо виникає потреба в такій зміні.

Проміжна функція придатності. Цей компонент служить метою функції придатності в будь-якій системі, заснованій на пошуку: він призначає кожному потенційному рішенню значення придатності. Різниця полягає в тому, що дозволяє вносити зміни до цього компонента під час процесу пошуку. Такі зміни виникають у результаті зворотного зв’язку, який надає людина-фахівець предметної галузі, і дозволяють їм впливати на напрямок автоматизованого пошуку, а також виконувати власний вибір.

Метою цього компонента є не замінити людський вхід, а скоріше надати початковий відбір кандидатів на рішення, щоб лише ті рішення, які, ймовірно, будуть найбільш успішними, аналізувалися людиною-фахівцем предметної галузі.

Взаємодія між внутрішнім і зовнішнім циклами буде здійснюватися через популяції кандидатів, оцінки, зроблені фахівцем предметної галузі, і зворотний зв’язок, який буде керувати проміжною функцією придатності. Генерація та відбір популяції, а також внутрішня робота внутрішнього циклу будуть приховані від фахівця предметної галузі.

Система, представлена тут, розроблена спеціально для ситуацій, коли знання предметної області є вирішальним фактором успішного тестування. Це може бути пов’язано зі складністю тестованої системи та впливом зовнішніх факторів на її роботу, а також обмеженнями щодо ресурсів, доступних для тестування.

В цьому розділі запропоновано систему тестування програмного забезпечення на основі пошуку, розроблену для того, щоб дозволити фахівцям домену з невеликим досвідом тестування програмного забезпечення розробляти тестові випадки для своїх програм. Цінність таких систем буде особливо актуальною в контекстах, де немає експертів з тестування програмного забезпечення або де знання предметної області є вирішальним фактором успіху процесу тестування.

Висновки до розділу

В першому розділі досліджується та формується теоретична базуадля розуміння процесів тестування програмного забезпечення, зокрема тих, що ґрунтуються на пошукових підходах.

У першому підрозділі було розглянуто основні принципи тестування програмного забезпечення, орієнтованого на пошук, яке використовує алгоритми оптимізації та евристики для автоматизованого створення тестових сценаріїв. Цей підхід забезпечує ефективність процесу тестування завдяки його здатності знаходити критичні помилки навіть у складних системах, де традиційні методи виявляються обмеженими.

Другий підрозділ присвячено аналізу інтерактивних систем тестування на основі пошуку. Такі системи поєднують автоматизацію пошукових методів із можливістю втручання тестувальника для уточнення критеріїв тестування, що дозволяє досягати гнучкості та точності у виявленні дефектів. Було зазначено, що інтерактивні системи забезпечують зменшення витрат часу на підготовку тестів та підвищують якість їх виконання завдяки адаптації до змінних умов розробки.

У третьому підрозділі обговорювалися основні концепції систем тестування на основі пошуку, включно з їх архітектурою, роллю метрик оцінки ефективності та способами інтеграції з процесами розробки. Особливу увагу приділено важливості використання багатокритеріальних

пошукових алгоритмів, які дозволяють оптимізувати тестування за кількома параметрами, такими як покриття коду, продуктивність системи та час виконання.

Загалом, у розділі було визначено ключові переваги тестування на основі пошуку: автоматизація рутинних задач, підвищення точності аналізу дефектів та адаптивність до складних і динамічних умов розробки. Ці аспекти забезпечують перспективність і важливість використання пошукових підходів для інтерактивного тестування сучасного програмного забезпечення.

РОЗДІЛ 2. МЕТОДИ ІНТЕРАКТИВНОГО ЕВОЛЮЦІЙНОГО ПОШУКУ ПІД ЧАС ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1. Особливості використання метаевристичних методів пошуку

Пошукове тестування програмного забезпечення використовує метаевристичні методи пошуку для автоматизації або часткової автоматизації завдань тестування, таких як генерація тестових випадків або генерація тестових даних. Воно використовує функцію пристосованості для кодування характеристик якості, які є релевантними для певної проблеми, і направляє пошук до прийнятних рішень у потенційно великому просторі пошуку.

З промислової точки зору, це відкриває можливість генерувати та оцінювати велику кількість тестових випадків без підвищення витрат до неприйнятного рівня. Однак спочатку необхідно оцінити застосовність пошукового інжинірингу програмного забезпечення в промисловому середовищі.

На практиці складно розробити a priori функцію пристосованості, яка охоплює всі практичні аспекти проблеми. Взаємодія з людськими експертами відкриває доступ до досвіду, який інакше недоступний, і дозволяє створювати більш обґрунтовану та точну функцію пристосованості.

В цьому розділі ми описуємо застосування інтерактивного пошукового тестування програмного забезпечення (ISBST) в промисловому середовищі. Ми використовували SBST для пошуку тестових випадків для промислового програмного модуля і частково спиралися на взаємодію з людиною-фахівцем галузі. Наша оцінка показала, що такий підхід є здійсненним, хоча він також виявив потенційні труднощі, пов’язані з взаємодією між фахівцем галузі та системою.

Програмне забезпечення часто розробляється як лише один компонент серед багатьох у складних інженерних системах. У такій ситуації не від усіх

розробників систем можна очікувати, що вони матимуть досвід в інженерії програмного забезпечення. Тим не менш, їхні доменні знання часто є критичними для створення та вибору тестових випадків. Пошукове тестування програмного забезпечення може автоматично створювати тестові випадки програмного забезпечення і, таким чином, потенційно залучати ширшу базу досвіду, представляючи ці тестові випадки розробникам і дозволяючи їм інтерактивно вибирати найбільш значущі з них [8, 9, 27]. Експерти-люди, звідси і звані «фахівцями галузі», є, таким чином, невід'ємною частиною розробки програмного забезпечення, використовуючи свої знання та досвід для прийняття таких компромісів.

Тести, розроблені таким чином, повинні бути представлені таким чином, щоб системні розробники, фахівці галузі та навіть користувачі могли зрозуміти і що дозволяє зробити обґрунтований вибір. Це можна досягти шляхом відповідності представлень тестів домену, а не до традиційних мов програмування та тестування. Таким чином, прийняття специфічних для домену представлень дозволяє представляти інформацію таким чином, що вже знайомий і уникнути додаткового тягаря адаптації до нових представлень.

У цьому розділі ми досліджуємо ефективність таких представлень для інтерактивного, напівавтоматизованого тестування програмного забезпечення для вбудованих систем управління, розроблених шведською компанією з інженерії систем і виробництва.

2.2. Існуючі підходи та методи інтерактивного еволюційного пошуку

Пошуковий інжиніринг програмного забезпечення (SBSE) є застосуванням метаевристичних методів пошуку до проблем інженерії програмного забезпечення. Важливою концепцією для систем на основі пошуку є поняття функції пристосованості. Функцію пристосованості можна

розглядати як «характеристику того, що вважається хорошим рішенням» [7]. Функція пристосованості використовується для вибору найкращих рішень у популяції та для направлення пошуку до хороших рішень.

В [10] автор визначає інтерактивне еволюційне обчислення як «ЕС, що оптимізує системи на основі суб'єктивної людської оцінки». Цей підхід спирається на людську взаємодію для оцінки рішень, що розробляються інтерактивною системою пошуку, дозволяючи ситуації, коли вибір рішення залежить від «людських уподобань, інтуїції, емоцій та психологічних аспектів». Оригінальна робота посилається на мистецтво та анімацію, графіку та обробку зображень; загалом на додатки, де оцінка кандидата має сильну суб'єктивну складову. Видатним прикладом цього є Picbreeder [32], онлайн-сервіс, де користувачі еволюціонують зображення в колаборативному середовищі, використовуючи інтерактивну еволюцію. Естетичні уподобання користувачів керують еволюцією, а не будь-яка об'єктивна мета.

Тим не менш, ми вважаємо, що це можна узагальнити до будь-якого типу оцінки кандидатів, де не вся необхідна інформація, наприклад, доменні знання, досвід, фонова інформація або інтуїція, може бути змодельована в системі або закодована в автоматичному підході до оцінки [12]. З точки зору системи, неявні знання можна розглядати як суб'єктивну оцінку, таким чином уникаючи необхідності дублювати існуючий досвід.

Автор в [10] також визначає проблему втоми людини. Це проблема, яка виникає, коли користувач-людина має виконувати велику кількість взаємодій із системою. Це проблема для будь-якої інтерактивної системи, оскільки людина, що страждає від втоми, не забезпечить рівень аналізу та прийняття рішень, необхідний для належного виконання своїх обов'язків. Тому ключовий елемент в інтерактивній системі не буде працювати належним чином і може навіть перешкоджати здатності системи еволюціонувати до хорошого рішення.

Підходи на основі пошуку вже використовувалися як дослідницькі інструменти в ситуаціях, коли існує неповне знання простору пошуку. Автор

в [12] описує використання генетичного програмування для дослідження потенційних конструкцій програмного забезпечення для системи арешту літаків і визначає важливість отримання проблемно-специфічних знань на ранніх етапах процесу проектування. В [39] вводять інтерактивну систему еволюційного проектування для підтримки ранніх етапів проектування. Вона визначає важливість захоплення «знань про дизайн через широку взаємодію дизайнера».

Системи SBST, такі як EvoSuite [33], вимагають досвіду інженерії програмного забезпечення для використання. Додавання інтерактивності до такої системи в нашому контексті вимагало б від фахівців галузі придбання додаткових навичок в інженерії програмного забезпечення, процес, який був би дорогим з точки зору часу і ресурсів.

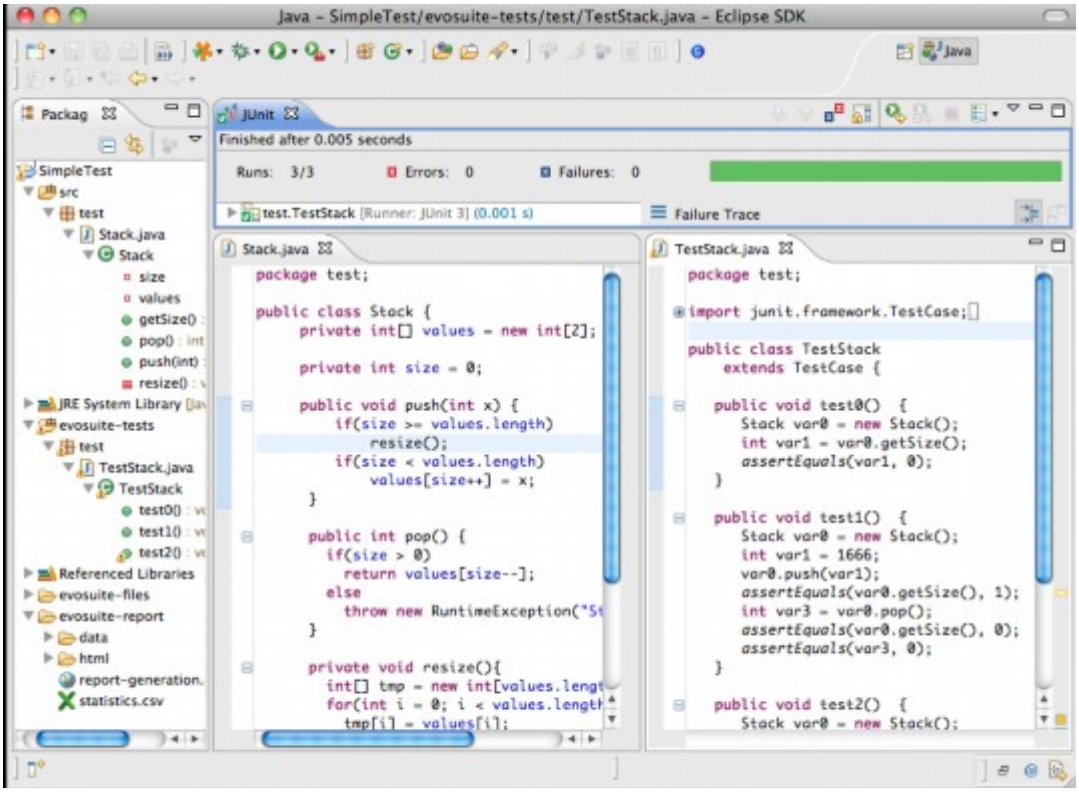


Рис. 2.1. Інтерфейс інструменту EvoSuite

Даний інструмент можна використовувати як інструмент командного рядка або як плагін Eclipse, повністю автоматично створюючи тестові набори з покриттям для класів Java.

EvoSuite — це інструмент, який автоматизує це завдання, систематично створюючи набори тестів, які досягають високого покриття коду, є якомога меншими і забезпечують твердження (див. рис. 2.1). EvoSuite використовує підхід на основі пошуку, інтегруючи такі передові методи, як гібридний пошук [9], динамічне символічне виконання [7] і трансформацію тестуваності [8]. Крім того, EvoSuite реалізує кілька нових методів для ефективного досягнення своїх цілей:

- Генерація цілого набору тестів: EvoSuite використовує еволюційний підхід до пошуку, який одночасно еволюціонує цілі набори тестів щодо всього критерію покриття [3]. Оптимізація щодо критерію покриття, а не окремих цілей покриття, досягає того, що результат не є несприятливо вплинутим ні порядком, ні складністю чи нездійсненністю окремих цілей покриття.

- Генерація тверджень на основі мутації: EvoSuite використовує мутаційне тестування для створення скороченого набору тверджень, що максимізує кількість виявлених дефектів у класі, виявлених тестовими випадками [5]. Ці твердження підкреслюють важливі аспекти поточної поведінки, щоб підтримати розробників у виявленні дефектів, і твердження фіксують поточну поведінку, щоб захистити від регресійних помилок.

EvoSuite повністю автоматично створює ці набори тестів для окремих класів або цілих проектів без необхідності складних ручних кроків. Інструмент доступний безкоштовно і може використовуватися в командному рядку, як плагін до платформи розробки Eclipse або через веб-інтерфейс.

EvoSuite - це інструмент, який автоматично створює набори тестів для Java-програм, що досягають високого рівня покриття коду та надають твердження. EvoSuite реалізує кілька нових методів, що призводить до вищого структурного покриття та ефективного вибору тверджень на основі засіяних дефектів, що є критичною особливістю, якої бракує іншим Java-інструментам.

Наразі EvoSuite підтримує покриття гілок та тестування мутацій як цілі тестування, але ми також працюємо над додаванням додаткових критеріїв, таких як критерії, засновані на потоці даних, а ще одним напрямком наших досліджень є створення більш зрозумілих тестових випадків.

Наша система відрізняється від попередніх підходів саме питанням використання поточного доменно-специфічного представлення як бази для взаємодії, одночасно намагаючись захистити фахівця галузі від специфічних деталей інженерії програмного забезпечення.

2.3. Методика застосування системи на практиці

Нехай1, певна компанія надає своїм клієнтам інструмент розробки програмного забезпечення, спеціально розроблений для того, щоб дозволити фахівцям галузі модифікувати та розробляти власне вбудоване програмне забезпечення для використання з універсальними контролерами.

Хоча точні додатки можуть сильно відрізнятися, у всіх цих випадках доменний досвід переважає важливий досвід в інженерії програмного забезпечення.

В результаті, хоча програмне забезпечення є важливим компонентом у їхніх відповідних продуктах, вони зосереджують свої зусилля та ресурси на інших компонентах.

У такому контексті якість отриманої системи залежить від якості програмного забезпечення, але не виключно. Можуть знадобитися компроміси, наприклад, обмеження можливостей програмного забезпечення через необхідність використання більш надійного та менш потужного обладнання, які не можуть бути відомі заздалегідь або можуть виникнути під час процесу проектування.

Фахівець галузі має знання про домен і досвід обмежень та вимог, що ставляться до систем, які вони розробляють. Ці знання дозволяють їм краще оцінювати характеристики якості повного продукту.

Практичним прикладом є механічна рука: гідравлічні клапани та електродвигуни керуються мікроконтролером. Програмне забезпечення для цього мікроконтролера часто розробляється ad-hoc для кожного застосування, тому не можна розробити узагальнені тестові випадки.

У цьому прикладі ми розробляємо тести для модуля програмного забезпечення мікроконтролера. Система, що тестується (SUT), є фільтром, який гарантує, що сигнал, наприклад, вхід для двигуна від користувача або інших модулів, не пошкоджує інші компоненти. Він робить це, гарантуючи, що сигнал не перевищує заданого верхнього ліміту і послаблюючи будь-які раптові зміни. Цей фільтр є відносно простим, але типовим компонентом: він достатньо поширений, щоб бути включеним у базову функціональну бібліотеку, яка надається з інструментом розробки, згаданим вище.

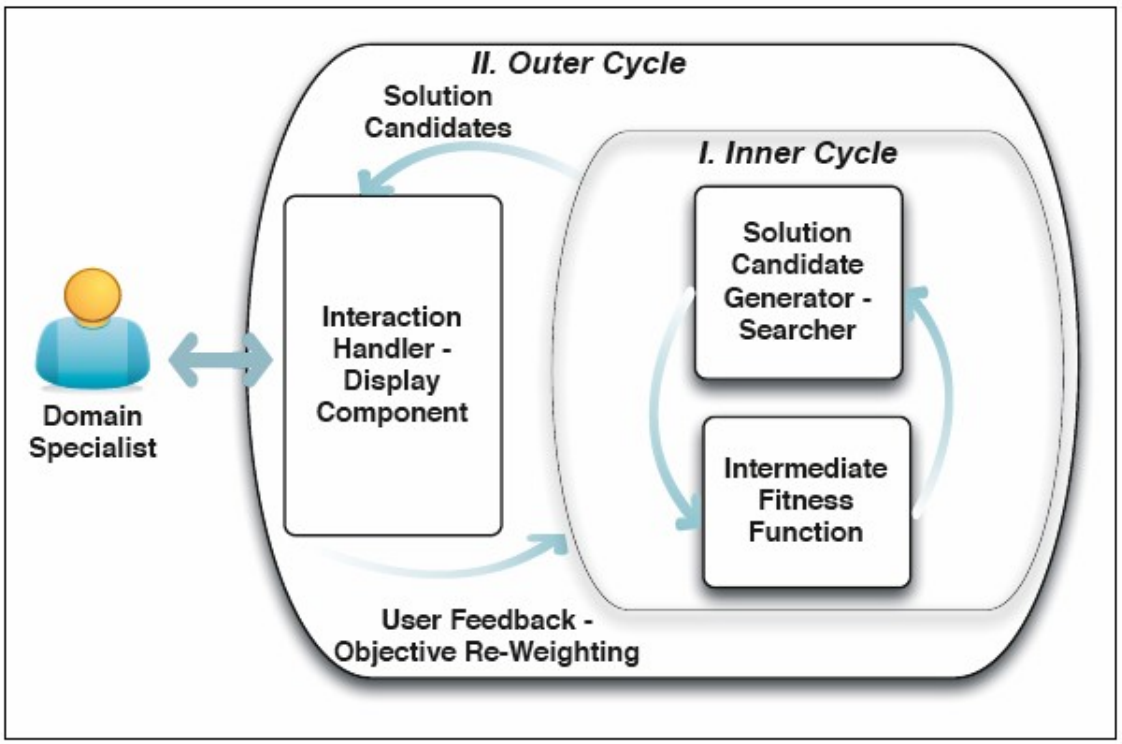


Рис. 2.2. Структура методики інтерактивного тестування

Рисунок 2.2 зображує циклічний процес, що складається з двох основних циклів: внутрішнього та зовнішнього. Ця структура часто використовується в системах, що базуються на еволюційних алгоритмах, для

вирішення задач оптимізації, проектування, машинного навчання та інших областей.

Внутрішній цикл

Генератор-пошукач рішень: Цей компонент відповідає за створення нових потенційних рішень або модифікацію існуючих. Він використовує різноманітні евристики або алгоритми пошуку (наприклад, генетичні алгоритми, імітаційний відпал) для генерації різноманітних варіантів.

Проміжна функція оцінки: Ця функція оцінює якість згенерованих рішень. Вона обчислює певну метрику (функцію придатності), яка відображає, наскільки добре дане рішення відповідає поставленій задачі.

Зовнішній цикл

Компонент взаємодії з користувачем: Цей компонент забезпечує взаємодію між системою та користувачем (доменним спеціалістом). Він відображає користувачу поточний стан системи, отримані результати та дозволяє користувачу вносити корективи в процес пошуку рішення.

Зворотний зв'язок користувача та переоцінка цілей: Користувач, аналізуючи результати роботи системи, може надати зворотний зв'язок, який використовується для коректування цілей пошуку. Це може виражатися у зміні ваг різних критеріїв оцінки рішень або у додаванні нових обмежень.

Загальна логіка роботи наступна.

Ініціалізація: Система починає роботу з деякого початкового набору рішень.

Внутрішній цикл:

Генерація: Генеруються нові рішення або модифікуються існуючі.

Оцінка: Кожне рішення оцінюється за допомогою проміжної функції оцінки.

Відбір: На основі отриманих оцінок відбираються найбільш перспективні рішення для подальшої роботи.

Зовнішній цикл:

Взаємодія з користувачем: Користувачу представляються результати роботи системи.

Зворотний зв'язок: Користувач аналізує результати і вносить необхідні корективи.

Переоцінка цілей: На основі зворотного зв'язку користувача корегуються цілі пошуку.

Повторення: Цикли повторюються до тих пір, поки не буде знайдено задовільне рішення або не буде досягнуто деякої заздалегідь визначеної умови зупинки.

Переваги такої структури:

Гнучкість: Можливість адаптуватися до змінних умов та уподобань користувача.

Ефективність: Використання евристичних методів дозволяє знаходити рішення у розумні терміни.

Інтерактивність: Взаємодія з користувачем дозволяє покращити якість отримуваних результатів.

Їхній підхід полягає в тому, щоб забезпечити фахівців галузі навчанням використання інструменту розробки програмного забезпечення, а не намагатися навчити інженерів програмного забезпечення доменним концепціям, які їм знадобляться. Інструмент розробки програмного забезпечення, який вони надають, використовує концепції, які вже знайомі тим, хто працює в цій галузі, наприклад, він виражає компонент з точки зору сигналів і операцій над сигналами, а не концепцій програмування.

Стажери, по суті, створюють типи моделей, які вони звикли розробляти, і ці моделі використовуються інструментом для генерації коду.

Ми досліджуємо інтерактивну систему пошукового тестування програмного забезпечення, адаптовану до конкретних потреб промислової компанії та їх інструменту розробки, і засновану на типі програмних додатків, які вони зазвичай розробляють.

2.4. Особливості побудови системи інтерактивного тестування програмного забезпечення

Система ISBST складається з системи тестування програмного забезпечення на основі пошуку, де спеціалісти домену можуть взаємодіяти та керувати пошуком тестових випадків для SUT. Фактичний SUT, що тестується, — це вихідний код, згенерований із візуальної моделі, яку практикуючий спеціаліст вказує за допомогою свого існуючого IDE; моделі схожі на функціональні блок-схеми, які використовуються при розробці вбудованих систем керування та програмного забезпечення. Потім система ISBST шукає тестові випадки, які порушують бажані вимоги SUT («не перевищувати цей ліміт вихідного сигналу» та «уникати різких розривів у вихідному сигналі», відповідно), зберігаючи тестові випадки якомога коротшими.

Взаємодія дозволяє фахівцям домену використовувати свій досвід для оцінки та ранжування рішень, розроблених системою ISBST. Щоб забезпечити цю взаємодію значущим чином, система використовує уявлення, які мають відношення до фахівця домену, комбіновані графіки вхідних і вихідних сигналів, а також таблиці пар вхідних і вихідних сигналів. Дотримуючись того способу, який зараз обдумують і визначають тестові випадки практики, зменшується прогалина у використанні системи ISBST. Цей тип представництва, специфічного для домену, відокремлює фахівця домену від дрібниць базової системи ISBST, наприклад, їм не потрібно піклуватися про те, як представляти тестові випадки, щоб вони були придатними для пошуку тощо.

Це означає, що спеціалісти, про яких йдеться, продовжуватимуть працювати зі знайомими їм концепціями та ідеями, і їм не потрібно буде розвивати спеціальні навички програмного забезпечення, щоб використовувати систему ISBST для створення тестових випадків.

Система ISBST (рис. 2.2) складається з двох вкладених циклів. Внутрішній цикл містить систему тестування програмного забезпечення на основі пошуку, яка є основою нашого підходу. Спеціаліст домену керує пошуком опосередковано, вибираючи критерії якості, які вони вважають важливими в даний момент часу, і визначаючи їх за пріоритетністю. Фітнес-функція потім адаптується відповідно до цих пріоритетів. Зовнішній цикл забезпечує взаємодію з фахівцем предметної області, включаючи вибір критеріїв якості та визначення пріоритетів, а також візуалізацію кандидатів.

На більш технічному рівні внутрішній цикл є серверною частиною системи та розроблений у Ruby. Він використовує алгоритм диференціальної еволюції, знайдений у бібліотеці FeldtRuby, щоб розробити кандидатські рішення, а потім варіант методу Bentley Sum of Weighted Global Ratios [50] для їх оцінки. Зовнішній цикл використовує комбінацію html і javascript для відображення рішень, які можна надати фахівцеві домену. Бібліотека документів, керованих даними (D3), щоб забезпечити відповідну графічну візуалізацію для потенційних рішень.

Як згадувалося вище, в [10] автор визначає втому користувача як основну проблему при використанні інтерактивних еволюційних обчислень. Він також пропонує деякі методи полегшення цієї проблеми.

У нашій системі зовнішній цикл вирішує будь-які питання щодо взаємодії з фахівцем домену, включаючи запобігання втоми користувача. З цією метою були використані деякі методи, представлені в [10]. Відображаються лише деякі з великої кількості рішень, що генеруються, причому вибір виконується на основі пріоритетів, визначених фахівцем домену. Події взаємодії відбуваються кожні 500 кроків оптимізації, щоб ще більше полегшити проблему втоми.

2.4.1. Використання прототипу системи тестування

Поточний прототип ISBST використовується через веб-інтерфейс. Це дозволяє фахівцю предметної області вибрати зі списку цілей ті, які

актуальні для нього в даний момент, і забезпечити пріоритетність, присвоївши відповідну вагу кожній меті.

Надане зважування використовується для динамічного розвитку функції проміжної придатності (IFF), яка буде використана ISBST для створення першого набору варіантів рішень.

Під час наступного етапу взаємодії фахівець домену може оцінити найвищі рішення з цього набору. Огляд потенційних рішень дозволяє порівняти з першого погляду (рис. 2.3), тоді як окремі перегляди можуть надати додаткову інформацію.

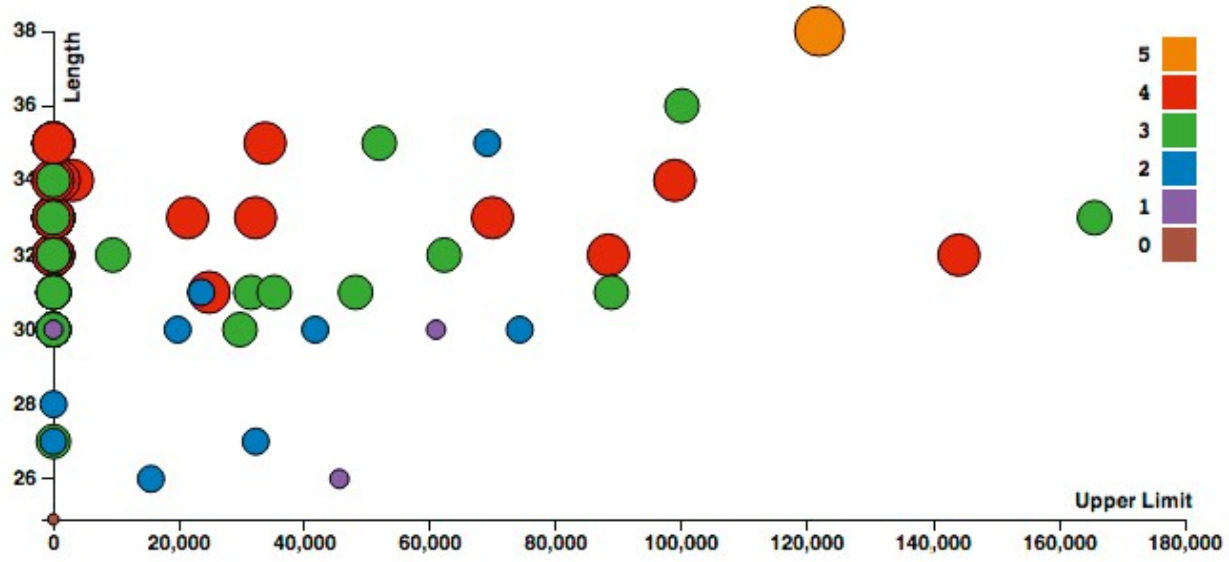


Рис. 2.3. Огляд найкращих доступних варіантів рішень під час взаємодії

Діаграма (рис. 2.3) показує інформацію щодо балів, отриманих кожним кандидатським рішенням щодо трьох об’єктів, які використовувалися для оцінки: Довжина тестового випадку (вісь Y), ступінь, до якого кожен тестовий приклад наближається до встановленої верхньої межі або навіть перевищує її. визначається верхньою межею (вісь X) і кількістю розривів, які були виявлені (колір). Ця діаграма надає швидкий спосіб порівняння кандидатів. Більше інформації про кожного кандидата також доступно у формі індивідуального перегляду.

Також під час етапу взаємодії фахівець із домену коригує свій вибір і пріоритезацію цілей шляхом їх повторного зважування. Після завершення всіх цих дій процес динамічного обчислення IFF, розробки нового набору рішень-кандидатів і представлення найкращих рішень для оцінки повторюється, доки не з’явиться одне або кілька задовільних рішень-кандидатів.

2.4.2. Емпірична оцінка прототипу

Емпірична оцінка прототипу має дві цілі: ширшу – дослідити застосовність у цій галузі промисловості та вужчу оцінку механізмів взаємодії, обраних для прототипу. Ці цілі є початковою оцінкою, частиною ширшої мети визначення рівня якості тестів, знайденого за допомогою цього методу. Огляд методу оцінювання можна побачити на рисунку 2.4.

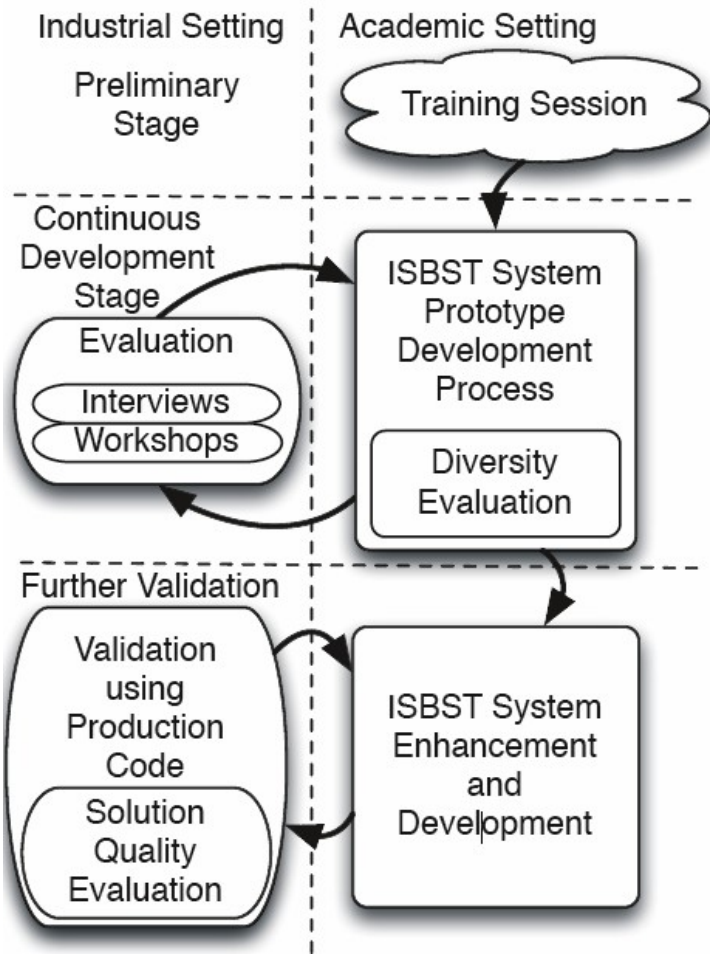


Рис. 2.4. Огляд етапів та емпіричної роботи

Першу мету, застосовність ISBST у цьому промисловому контексті, оцінювали за ступенем, до якого прототип системи розвивав рішення відповідно до цілей пошуку та вагових коефіцієнтів, встановлених фахівцем у галузі.

Друга мета, оцінка та вдосконалення механізму взаємодії, який використовується прототипом системи ISBST. Механізм полягає в динамічному розвитку функції проміжної придатності (IFF) на основі об'єктивного відбору та повторного зважування. Щоб оцінити цей механізм, ми вимірюємо ступінь, до якого вибір функції відповідності впливає на різноманітність отриманих тестових випадків.

Емпірична оцінка складалася з двох етапів. На першому етапі вибір механізму взаємодії перевіряли в лабораторних умовах. Другий етап полягав у оцінці прототипу ISBST спільно з командою розробки та тестування. На практиці ці два етапи збігалися в етап безперервного розвитку (рис. 2.3), з новою інформацією, яка включалася в систему.

У таблиці 2.1 показано кілька стратегій використання, які досліджувалися в рамках емпіричної оцінки. Вплив кожної зі стратегій на різноманітність популяції кандидатів можна побачити на рисунку 2.5.

Таблиця 2.1.

Стратегії взаємодії, які використовуються для дослідження збереження різноманітності популяції

Scenario	Description
A	Non-dynamic, multi-objective fitness function.
B	Focus in on one single objective that completely and consistently outweighs the others.
C	The objective under focus outweighs the others, but is changed at the last step.
D	A more balanced approach. One objective outweighs the others, but it is not the sole focus of attention.

Другий етап включав дослідження та тестування, що забезпечило подальшу перевірку механізму взаємодії.

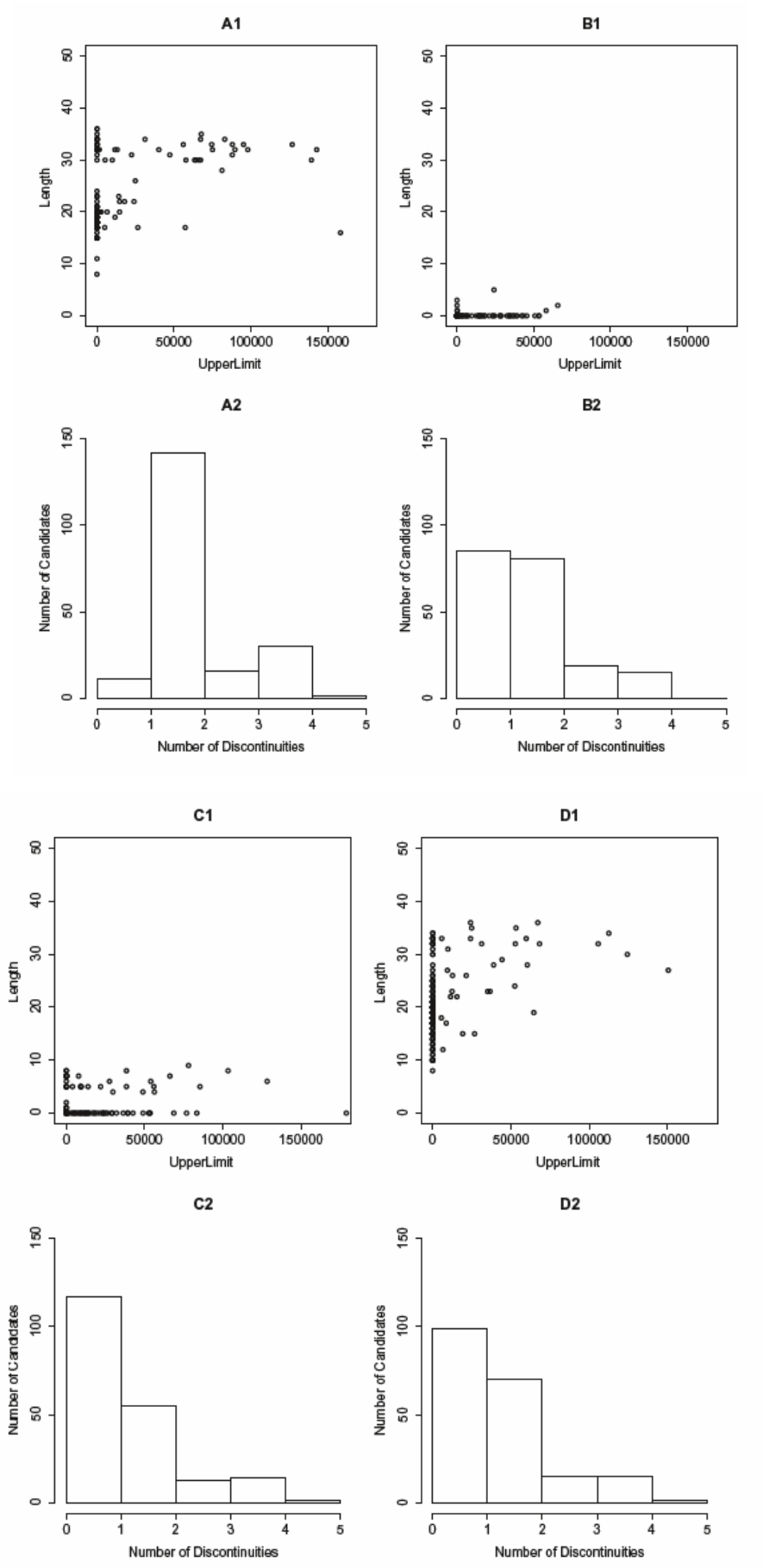


Рис. 2.5. Різноманітність популяції після набору інтерактивних цілей

Результати емпіричних досліджень описані в таблиці 2.2.

Таблиця 2.2.

Результати емпіричного оцінювання

	Description
1	**Understandability**. Providing clear, understandable, and accurate information to the domain specialist is the key element in establishing a meaningful interaction between them and the ISBST system.
2	**Variation**. A single developer may work on several systems in a short amount of time. In addition, there is a large amount of variation in terms of the types of systems developed within the same company.
3	**Individual Preferences**. Even when faced with similar systems, different individual domain specialists may have different preferences regarding how the interaction should take place and what information should be made available.
4	**Dynamic Validation**. Dynamically validating the system allowed us to identify potential problems in a timely manner and to develop a system that was relevant to our industrial partners and that could, in turn, benefit from their experiences.

Загалом результати вказують на те, що механізми, які ми використовували для взаємодії між фахівцем домену та прототипом ISBST, корисні та придатні для використання. Спеціалісти домену добре відреагували на прототип і були надзвичайно корисними як в його оцінці, так і в наданні пропозицій щодо вдосконалення в наступних версіях.

На більш практичному рівні ряд згенерованих тестів успішно досягли цілей пошуку, тобто виявлення вхідних даних, які викликають перевищення системою максимального встановленого значення або призводять до розривів вихідного сигналу. Для майбутньої роботи однією із середньострокових цілей є використання знань і досвіду спеціалістів у галузі, щоб спрямувати систему ISBST на цікаві та значущі тестові випадки, таким чином підвищуючи якість без надмірних витрат. Інший полягає в застосуванні цієї концепції в інших областях, які можуть отримати вигоду від автоматизованої розробки тестів, але де знання предметної області є життєво важливими. Далі

тим не менш, інші етапи процесу розробки програмного забезпечення також можуть виграти від поєднання людського розуміння та автоматизованих обчислювальних потужностей.

Однак на додаток до цього бачення слід також обговорити деякі загрози достовірності цього дослідження. Прототип системи ISBST був розроблений для конкретної компанії і тому обмежений використанням єдиного набору процедур і опорою на одне джерело інформації та досвіду. Тим не менш, компанія розробляє широкий спектр вбудованого програмного забезпечення для контролерів загального типу. У майбутніх дослідженнях питання можливості узагальнення буде розглянуто далі, але, виходячи з поточних результатів, ми вважаємо, що цей підхід має значні перспективи.

Отже, було представлено практичне застосування алгоритмів на основі пошуку. Прототип системи тестування програмного забезпечення на основі інтерактивного пошуку (ISBST), який ми запропонували та впровадили, використовує знання та досвід спеціалістів у галузі, щоб скеровувати пошукові алгоритми до цікавих рішень.

Початкові результати є багатообіцяючими, показуючи, що взаємодія між фахівцями домену та автоматизованими інструментами для створення тестів є надійним підходом і може дати корисні результати. Ті самі результати також показують важливість механізмів взаємодії та інформації, яка надається фахівцям домену для прийняття рішень. Очевидна важливість динамічної перевірки за участю персоналу компанії та в промисловому контексті, оскільки практичне використання прототипу дало більше інформації, ніж попередні спроби статичної перевірки.

Висновки до розділу

Даний розділ присвячено аналізу та розробці методів інтерактивного еволюційного пошуку, що використовуються в процесі тестування програмного забезпечення. У ході дослідження були зроблені такі висновки:

Метаевристичні методи пошуку мають важливе значення для вирішення задач тестування програмного забезпечення. Вони забезпечують ефективність процесу пошуку рішень завдяки можливості адаптації до складних та динамічних умов. Використання таких підходів дозволяє оптимізувати тестування шляхом зменшення кількості тестових випадків при збереженні високого рівня виявлення дефектів.

Було детально розглянуто існуючі підходи до інтерактивного еволюційного пошуку. Зокрема, визначено переваги і недоліки різних алгоритмів та технік, що застосовуються для пошуку оптимальних рішень у контексті тестування програмних систем. Це дозволило визначити ключові аспекти, які потребують удосконалення, зокрема інтерактивність і точність пошуку.

Розглянуто особливості побудови системи інтерактивного тестування, де акцент зроблено на використанні прототипу як інструменту для оцінки можливостей системи. Прототип дозволяє не лише протестувати окремі компоненти, але й оцінити інтегровану функціональність.

Проведена емпірична оцінка прототипу підтвердила його здатність виконувати ключові задачі інтерактивного еволюційного пошуку. Це свідчить про доцільність подальшого розвитку та впровадження таких систем у процеси тестування складних програмних рішень.

Таким чином, у розділі визначено як теоретичну, так і практичну цінність інтерактивного еволюційного пошуку. Отримані результати створюють базу для подальшого вдосконалення систем тестування, спрямованих на підвищення якості та надійності програмного забезпечення.

РОЗДІЛ 3. ТЕХНІКА ТА МЕТОДОЛОГІЯ ОЦІНКИ ІНТЕРАКТИВНОГО ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Важливість показника якості програмного забезпечення

Програмне забезпечення, особливо вбудоване програмне забезпечення, є невід'ємною частиною різноманітних складних систем, які використовуються в багатьох галузях. Компанії, що розробляють такі системи, зосереджуються на своїх основних компетенціях у галузевих знаннях та досвіді, а не на інженерії програмного забезпечення та тестуванні програмного забезпечення. В результаті вони часто не мають достатнього досвіду для проведення систематичного тестування програмного забезпечення та контролю якості, зосереджуючись замість цього на тестуванні продукту в цілому. Оскільки якість розроблених продуктів залежить від низки компромісів, забезпечення якості програмного забезпечення часто не є пріоритетним завданням. Розвиток внутрішнього досвіду в галузі програмного забезпечення є надмірно дорогим, і компанії часто віддають перевагу зосередженню своїх ресурсів на покращенні конкурентних переваг у конкретній галузі.

Тому важливо дати змогу фахівцям у галузі покращувати якість розробленого ними програмного забезпечення, не відволікаючи їх від основних завдань. Це можна досягти шляхом розробки попередньо упакованого набору інструментів для тестування програмного забезпечення, який би пропонував передові практики розробки програмного забезпечення без необхідності опанування деталей, що лежать в основі інструменту. Ця концепція є розумною, але розробка такого пакету до того, як будуть відомі специфічні особливості застосування, є складним завданням. Більше того, функціональність додатків і способи їх тестування можуть змінюватися або відрізнятися між різними тестувальниками та фахівцями галузі, що ще

більше підкреслює важливість можливості використовувати галузеві знання як невід'ємну частину процесу тестування та мати гнучкий інструмент, який може адаптуватися до різних сценаріїв і типів використання.

Цей розділ пропонує систему для тестування вбудованого програмного забезпечення шляхом застосування техніки, яка в значній мірі автоматизує генерацію тестових даних, одночасно дозволяючи фахівцям у галузі вносити свої знання та досвід, тим самим дозволяючи їм зосередитися на галузевих проблемах. Застосована автоматизована техніка використовує метаевристичний оптимізаційний алгоритм для генерації тестових даних і, таким чином, є формою пошукового тестування програмного забезпечення [8, 9]. Фахівець у галузі взаємодіє з системою, щоб направляти оптимізаційний алгоритм у генерації тестових випадків, які є доречними в даному контексті. Ця взаємодія побудована на основі існуючої роботи в галузі інтерактивних еволюційних обчислень [10, 12, 27, 28, 36] і розроблена таким чином, щоб полегшити фахівцям у галузі внесення свого внеску, одночасно захищаючи їх від деталей реалізації самого інструменту.

Пропозиція про те, що пошукове тестування програмного забезпечення може бути поєднане з взаємодією користувача з метою ефективної генерації тестових випадків користувачами, які не обов'язково є експертами з тестування. Промислова оцінка, яка демонструє, що систему ISBST можуть успішно використовувати фахівці галузі для розробки тестових випадків без необхідності широкого навчання з її використання. Лабораторний експеримент, який підтверджує внесок базового алгоритму генерації тестів на основі пошуку.

3.2. Існуючі підходи до інтерактивного еволюційного пошуку

Пошукова інженерія програмного забезпечення (SBSE - Search-Based Software Engineering) — це термін, введений в [7] для опису застосування алгоритмів метаевристичної оптимізації (або «пошуку») до проблем

інженерії програмного забезпечення. Галузь SBSE, яка займається проблемами тестування, відома як тестування програмного забезпечення на основі пошуку (SBST) і застосовувалася до багатьох типів проблем тестування [8, 9], від об'єктно-орієнтованих контейнерів [24] до динамічних мов програмування [25].

Передумова SBSE полягає в тому, що для багатьох проблем інженерії програмного забезпечення важко отримати рішення напряму, але часто легко перевірити, чи даний «кандидат» рішення вирішує проблему. У контексті SBST зазвичай проблема полягає в отриманні тестових даних, які задовольняють конкретну мету тестування: хоча може бути важко отримати відповідний тестовий приклад, якщо нам надають кандидатський тестовий приклад, зазвичай легко перевірити, чи відповідає він мета тестування. Якщо можна визначити функцію пристосованості, яка вимірює ступінь, до якого рішення-кандидат вирішує проблему, тоді можна використовувати цю функцію пристосування, щоб направляти метаевристичний алгоритм оптимізації до рішень, які вирішують проблему. Навіть незважаючи на те, що алгоритму оптимізації може знадобитися побудувати та оцінити велику кількість потенційних рішень, щоб знайти те, яке вирішить проблему, цей підхід часто менш витратний, ніж розв'язання тієї ж інженерної проблеми вручну. Багато метаевристичних алгоритмів оптимізації працюють із сукупністю, тобто набором індивідуальних варіантів рішення, і такий алгоритм використовується в системі ISBST, описаній в цьому розділі. З цієї причини ми будемо взаємозамінно використовувати терміни «можливе рішення», «кандидат» і «індивідуальний» для позначення потенційного рішення, розробленого системою на основі пошуку.

Було проведено порівняно небагато досліджень щодо інтерактивного SBSE або SBST. В роботі [27] описано інтерактивне середовище розробки, де тести створюються, коли інженер пише програмний код або вдосконалює специфікацію. Система використовувала взаємодію інженера, щоб допомогти керувати пошуком, але вплив на функцію придатності був непрямим.

Прототип реалізації WISE був реалізований на мові об’єктно-орієнтованого програмування Ruby. Він називається WiseR (Wise for Ruby). Ruby був обраний, оскільки це мова високого рівня з багатьма функціями, які підтримують швидке прототипування. Він належить до нового класу мов, що підтримують динамічну типізацію та легкий доступ до всіх частин середовища виконання. Це вважалося необхідним для проведення експериментів з різними частинами системи.

На відміну від популярних мов Java і C++, Ruby є об’єктно-орієнтованою. На відміну від них, він має динамічну типізацію, тобто перевірка типів не проводиться під час компіляції. Насправді, компіляції немає, оскільки Ruby не компілюється, а інтерпретується.

Незважаючи на те, що ці аспекти роблять Ruby нетиповим порівняно з основними мовами, що використовуються сьогодні, ми не вважаємо, що це плутає наші результати. Навпаки, динамічна типізація ускладнює роботу тестувальника. Він не може просто подивитися на код і побачити, які типи дозволені для параметрів. В певному сенсі, в коді доступно менше інформації, і тому більше інформації потрібно повторно відкривати. Наявність компілятора прискорила б систему, зробила клієнта більш чутливим і дозволила б більш потужні обчислення.

WiseR реалізований на Ruby і підтримує розробку коду на Ruby. Усі три артефакти виражаються як код Ruby. Хоча це не є вимогою в WISE, це полегшує роботу з нашим прототипом. Ми можемо повторно використовувати загальні функції, що використовуються для аналізу артефактів. Також легше обмінюватися інформацією між ними.

Під час розробки WiseR основна увага приділялася модулю пошуку тестів, WiseR-Tests. Крім WiseR-Tests, система містить графічний інтерфейс користувача та невеликі реалізації модуля керування та бази знань. Головне вікно графічного інтерфейсу користувача WiseR показано на рисунку 3.1.

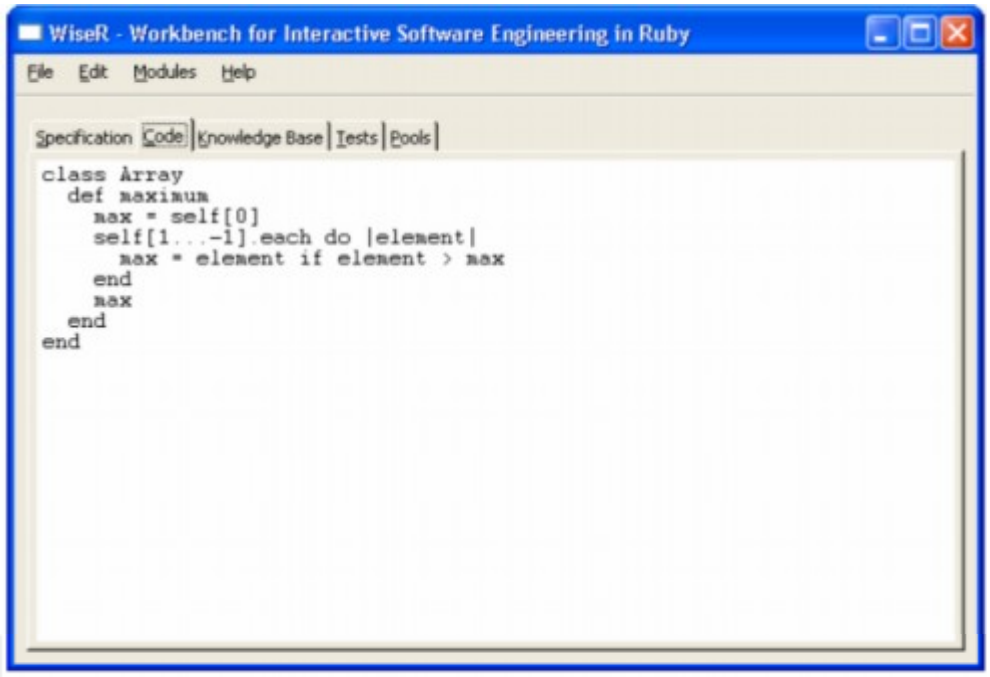


Рис. 3.1. Графічне вікно WiseR з активним вікном коду, завантаженим вихідним кодом Array#maximum

Інші роботи [13] і [39], розглядають використання інтерактивного пошуку для вивчення інженерних проектів і кращого розуміння обмежень дизайну, але не зосереджувався безпосередньо на тестуванні програмного забезпечення.

Тим не менш, поняття інтерактивної участі в процесі пошуку є усталеним. В [10] опису.nm Interactive Evolutionary Computation (IEC) як «EC, який оптимізує системи на основі суб'єктивної людської оцінки». Цей підхід використовує людину як замінну функцію придатності в ситуаціях, коли мета оптимізації залежить від «людських переваг, інтуїції, емоцій і психологічних аспектів» [10]; це включає такі програми, як мистецтво та анімація, комп'ютерна графіка та обробка зображень.

Також в [10] визначають три основні підходи до взаємодії людини з системою еволюційного обчислення (EC). По-перше, людина може діяти як звичайна фітнес-функція: людині представлено набір кандидатів, і вона повинна призначити оцінку фітнесу кожному з них. Це означає, що кожен кандидат повинен бути проаналізований і оцінений.

Другий підхід полягає в тому, щоб представити людині кандидатів для оцінки; Потім людина вибирає тих, хто є видатним, або вибираючи «хороших» кандидатів для просування до наступного покоління, або вибираючи «поганих» для виключення. У цьому підході лише підмножина кандидатів повинна бути відзначена, ранжована або оцінена. Це допомагає керувати пошуком, гарантуючи, що бажані характеристики завжди представлені в популяції та мають вищі шанси на поширення в наступному поколінні. По суті, користувач керує пошуком, вибираючи тих кандидатів, які вважаються «найкращим поточним представленням цілі» [37].

Третій визначений підхід — візуалізований ЕС, де людина вибирає рішення на основі значень придатності для кількох цілей, а не аналізує самих окремих кандидатів. Більш детально цей підхід описано в [38]. Один із таких прикладів представлений в [39], де кандидатом рішення є запропонований розподіл програмних компонентів у кластери. Замість того, щоб оцінювати кожного кандидата самостійно, користувач повинен вирішити, чи належать два компоненти до одного кластеру чи ні.

Ще одна додаткова концепція, пов’язана з IEC, — це гіперінтерактивність, яка визначається як «форма IEC, у якій користувач-людина активно вибирає, коли і як застосувати кожен із доступних еволюційних операторів, відіграючи центральну роль у потоці керування еволюційним процесом. пошукові процеси» [36]. Тут людина діє як прямий провідник у розвитку кожного бакалавра, а не як замінна функція придатності, оцінюючи кандидатів після того, як вони були створені.

Усі ці підходи вимагають, щоб людина взаємодіяла з системою принаймні один раз у кожному поколінні. Перші два підходи означають, що людина повинна оцінити кожного кандидата, перш ніж призначати оцінки фізичної підготовки або робити свій вибір. Ці особи-кандидати можуть бути досить складними конструкціями, що призводить до труднощів у проведенні послідовних, неупереджених і точних оцінок.

Коли людина більшою чи меншою мірою аналізує кожного кандидата, кількість кандидатів, які можна обробити, зменшується. Проблема втоми від прийняття рішень уже виявлена, і докладено зусиль, щоб її вирішити або, принаймні, мінімізувати її вплив [41].

Одним із запропонованих способів вирішення проблеми людської втоми, особливо втоми від прийняття рішень, є пошук способу вимірювання фізичної форми, який не вимагає від користувача такої частої взаємодії з системою. Наприклад в [44], описано систему, яка вимагає введення даних від користувача лише тоді, коли існуюча пріоритезація фітнес-функцій призводить до рівності, тим самим зменшуючи вимоги до користувача-людини.

Візуалізований підхід EC (описаний вище) вимагає лише, щоб людина оцінила оцінки придатності, які вже отримав кожен кандидат, а не саме рішення кандидата. Однак це все одно означає, що кожне рішення-кандидат потрібно розглядати та порівнювати з іншими. Гіперінтерактивність означає, що людина-користувач має бути надзвичайно залученим не лише до оцінки кожного кандидата, але й до розробки кандидатів із застосуванням еволюційних операторів.

Альтернативний підхід, який суттєво відрізняється від описаних досі, полягає в розробці сурогатних функцій фітнесу, спрямованих на заміну поведінки людини [45]. Хоча цей підхід схожий на нашу мету, ключова відмінність полягає в тому, що критерії якості, які ми досліджуємо, можуть різко змінюватися, коли стає доступною нова інформація. У нашому контексті суб’єктивна оцінка людини — це спосіб для фахівців у галузі поділитися своїми знаннями та досвідом, зосереджуючись при цьому на об’єктивних критеріях якості, а не на естетичних.

Ближче до нашого контексту є робота [47], де користувач обирає бажане рішення, а система повторно зважує цільові показники якості, доки рішення, якому віддає перевагу користувач, не матиме найвищої придатності. Навпаки, наша мета полягає в тому, щоб активно працювати над тим, щоб

функція фітнесу тісно відповідала поточному розумінню пріоритетів і відносної важливості цілей якості, які має фахівець у галузі. Таким чином, навіть якщо кандидати, які наразі відображаються, не демонструють якостей, необхідних спеціалісту домену, вони все одно можуть керувати пошуком відповідно до своєї оцінки того, яким критеріям якості повинна відповідати система.

В роботі [48] поєднують обчислювальну продуктивність із перевагою особи, що приймає рішення, щоб покращити процес вибору концептуального проекту. Проте їхня робота зосереджена на проблемі, продуктивність якої можна об’єктивно обчислити, а критерії для цього вже існують. Вплив особи, яка приймає рішення, в ідеалі полягає у виборі між проектами, порівнянними за якістю. В [49], подібним чином використовують переваги користувача як вхідні дані, у «поступово інтерактивний» спосіб, для набору недомінованих точок. У їхній статті йдеться про те, що існує набір об’єктивних заходів для визначення домінування та набір критеріїв, за якими оцінюється домінування. Навпаки, наша робота спрямована на дослідження простору тестових випадків і пошуку «цікавих», тобто невідомих раніше, тестових випадків і поведінки. У цьому контексті важко визначити об’єктивний показник ефективності, а критерії того, що таке цікаве рішення, є мінливими.

Автори в [50] визначають елегантність як ключовий фактор у проектуванні програмного забезпечення. Вони визначають набір кількісних показників елегантності, і їхня оцінка придатності бере один із цих показників випадковим чином і показує користувачеві найбільш елегантне рішення відповідно до цього показника. Ці показники є досить специфічними для домену та визначення елегантності авторів. У нашому контексті розуміння фахівцями предметної галузі того , що є хорошим рішенням, може змінюватися з часом або не піддається кількісному виміру.

В статті [46] запропонували систему, яка поєднує декілька з цих концепцій, наприклад, взаємодія із системою один раз у декілька поколінь , а не кожне покоління; і додає взаємодію із системою ISBST, дозволяючи

людині динамічно змінювати функцію пристосованості під час процесу пошуку, коли її розуміння змінюється або стає більш витонченим, або коли нова інформація стає доступною. В цьому розділі ми розширюємо цю роботу та оцінити в промислових умовах як корисність підходу ISBST, так і те, як користувачі взаємодіють із процесом пошуку.

3.3. Концепція побудови системи інтерактивного тестування програмної архітектури

Хоча загальний підхід ISBST має широке застосування, цей опис також охоплює деталі, специфічні для реалізації. Зокрема, зразкова система, що тестується (SUT), є SoftRamp, загальним компонентом у наборі інструментів DomainDevEnvironment, який використовується і функціональність якого була описана вище; а набір цілей якості є специфічним для типу програмного забезпечення, яке розробляється за допомогою набору інструментів.

Система ISBST розроблена таким чином, щоб полегшити фахівцям у галузі взаємодію та внесок своїх знань і досвіду, одночасно захищаючи їх від дрібних деталей базової пошукової системи.

Для досягнення цієї мети систему можна уявити як два вкладені цикли, як показано на рисунку 3.2. Внутрішній цикл використовує алгоритм на основі пошуку для створення популяції кандидатів на рішення. Зовнішній цикл займається взаємодією з фахівцем у галузі та перетворенням його відгуків на відповідну функцію пристосованості, яка керує пошуковим алгоритмом внутрішнього циклу. Система підтримує загальну структуру, представлену в попередньому розділі, враховуючи зростаючу складність.

3.3.1. Внутрішній цикл роботи системи

Внутрішній цикл містить усі технічні елементи, необхідні для розробки кандидатів на рішення та взаємодії з SUT для оцінки цілей якості.

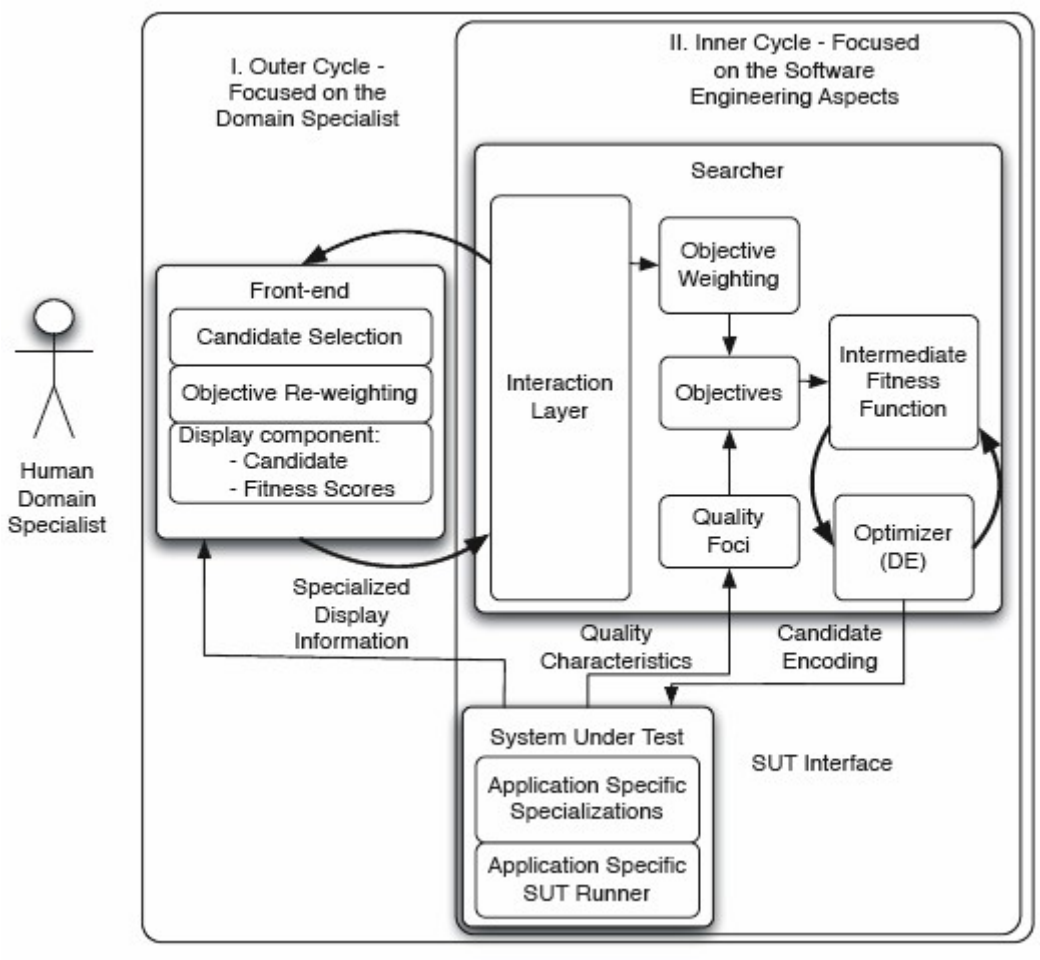


Рис. 3.2. Огляд системи інтерактивного тестування програмної архітектури

Кодування рішення. Кандидат на рішення в нашому контексті є тестовим випадком, що складається з набору входів, згенерованих алгоритмом на основі пошуку.

Для компонента SoftRamp входи поділяються на значення налаштування та вхідний сигнал. Значення налаштування - це набір з п’яти цілих чисел, які визначають налаштування SoftRamp. Деякі з цих значень підпадають під спеціальні правила, наприклад, два значення, sftstrt і sftend, є відсотками, і sftstrt + sftend 100. Ці значення генеруються випадковим чином із допустимих значень. Вхідний сигнал може складатися з будь-якого типу даних, підтримуваного DomainDevEnvironment. SUT, що використовується в даний час, приймає вхідний сигнал із фіксованою довжиною 15 значень, кожне з яких є 16-бітовим цілим числом.

Отже, кандидат на рішення є вектором з 20 дійсних чисел, перші 5 з яких є значеннями налаштування для SUT, а решта 15 є вхідним сигналом для SUT. Для кожного кандидата на рішення генерується вихідний сигнал шляхом запуску SUT з 5 значеннями налаштування та збору вихідного сигналу, що відповідає входу.

Цілі якості. Під час дослідження ми визначили ряд «цілей якості», тобто характеристик тестового випадку, які вказують на проблему в SUT і, таким чином, роблять хороший тестовий випадок. Ці цілі якості, більш детально описані в таблиці 3.1, обчислюються з результатів роботи SUT при запуску з тестовим випадком-кандидатом.

Таблиця 3.1.

Огляд поточних цілей якості

No.	Objective	Description	Aim
1	Large Signal at Least Once	Rewards higher maxima values in the output	Maximize
2	Small Signal at Least Once	Rewards lower minima in the output.	Minimize
3	Mean Output	Rewards higher mean values in the output.	Maximize
4	Above Important Limit at Least Once	Rewards test cases with output values that exceed by the most a prescribed maximum value. Only the largest difference is considered.	Maximize
5	Below Important Limit at Least Once	Rewards test cases with output values that fall farthest below a prescribed minimum value. Only the largest difference is considered.	Maximize
6	Above Important Limit Overall	Rewards test cases with mean output values that exceed by the most a prescribed maximum value.	Maximize
7	Below Important Limit Overall	Rewards test cases with mean output values that fall farthest below a prescribed minimum value.	Maximize
8	One Large Increase	Large variations in the output signal may damaging components, or be indicative of internal faults in the module. Rewards the maximum value of the first order derivative.	Maximize
9	One Large Decrease	Large variations in the output signal may damaging components, or be indicative of internal faults in the module. Rewards the minimum value of the first order derivative.	Minimize
10	Swings Through Zero	Oscillations in the output may be damaging to the other components or indicative of internal faults. Rewards the highest number of times the output signal crossed the 0 value.	Maximize
11	Diversity	This is an overall measurement of how different a test case is from the population in the previous interaction event. This allows the domain specialist to widen the search space being explored or converge towards a solution.	Maximize

Проміжна функція пристосованості. Механізм, який ми обрали, щоб дозволити фахівцям у галузі керувати пошуком, полягає в кодуванні їхнього відгуку щодо відносної важливості цілей якості в проміжній функції пристосованості (IFF). Ця функція обчислюється з набору балів за цілями якості для кандидата та набору ваг для цих цілей і є варіантом суми зважених глобальних відношень Бентлі [50]. Цей підхід нормалізує всі значення в поколінні до інтервалу між найбільшим і найменшим значеннями, спостереженими для даної мети, як у поточному, так і в попередніх поколіннях. Кожне рішення оцінюється і отримує бал за кожну з цілей якості. Потім ваги використовуються для об'єднання балів в одне значення пристосованості для кожного кандидата.

$$IFF(j) = \sum_{i=1}^{\text{nObjectives}} \text{Weight}_i * \text{Value}_{i,j}$$

Де IFF(j) є значенням пристосованості кандидата j, Weight_i є поточною вагою мети i, а Value_i,j є значенням пристосованості кандидата j, виміряним за метою i. Значення IFF(j) є сумою зважених значень пристосованості для всіх nObjective цілей. Ціллю k можна відмовитися від обчислення, встановивши Weight_k = 0.

Ваги отримуються від зовнішнього циклу. Ваги змінюються відповідно до відгуку, отриманого від фахівця галузі, і IFF перераховується, щоб відобразити ці зміни. Більш детальну інформацію про цілі якості та їх зв'язок із взаємодією користувача можна знайти в підрозділі нижче.

3.3.2. Алгоритм пошуку

Алгоритмом на основі пошуку, обраним для цієї реалізації, був Диференціальний еволюційний алгоритм (DE) [49]. Диференціальна еволюція є паралельним методом прямого пошуку. Кожне потенційне рішення є вектором дійсних чисел. Початкова популяція вибирається

випадковим чином з рівномірного розподілу і охоплює весь параметричний простір. Нові вектори параметрів додаються шляхом мутації: додавання зваженої різниці між двома векторами популяції до третього вектора. Для кожного цільового вектора xi,G, де i = 1, 2, ..., NP, вектор мутанта генерується наступним чином:

$$v_{i,G+1} = x_{r1,G} + F * (x_{r2,G} - x_{r3,G})$$

де r1, r2, r3 є цілими числами з діапазону 1, 2, ..., NP, взаємно різними і відмінними від поточного індексу i. F є дійсним і постійним коефіцієнтом з діапазону (0, 2], який контролює посилення диференціальної варіації (xr2,G - xr3,G).

Результат vi,G+1 потім піддається схрещуванню шляхом змішування його параметрів з параметрами іншого попередньо визначеного вектора, а результат цієї операції називається пробним вектором. Якщо пробний вектор є покращенням порівняно з цільовим вектором, він замінює його в наступному поколінні [49].

Швидкість схрещування, яку ми використовували, становить cr = 0,5, коефіцієнт масштабування становить F = 0,7, а розмір популяції становить population = 100. Стратегія мутації є тією, що запропонована в [49]: DE/rand/1/bin. Стратегія використовує алгоритм диференціальної еволюції (DE); вектор для мутації вибирається випадковим чином (rand); використовується один вектор різниці (1); схема схрещування є біноміальною (bin).

3.3.3 Зовнішній цикл

Зовнішній цикл є компонентом, відповідальним за обробку взаємодії з фахівцем галузі. Після кількох кроків оптимізації, steps = 300, внутрішній цикл зупиняє пошук і ініціює «подія взаємодії». Ця подія складається з: а)

відображення поточного та попередніх поколінь фахівцю галузі; б) відображення додаткових деталей за запитом; і в) дозволу фахівцю керувати тим, як продовжити пошук.

Поточні та попередні покоління відображаються у вигляді графіків, а осі графіків показують значення пристосованості, отримані кожним кандидатом на рішення щодо вибраних цілей якості, як показано на малюнку 3.3. Щоб візуалізувати більше, ніж пару цілей якості, фахівець галузі може вибрати перегляд матриці графіків, де кожен графік у матриці показує значення пристосованості пари вибраних цілей якості, наприклад, рисунок 3.4.

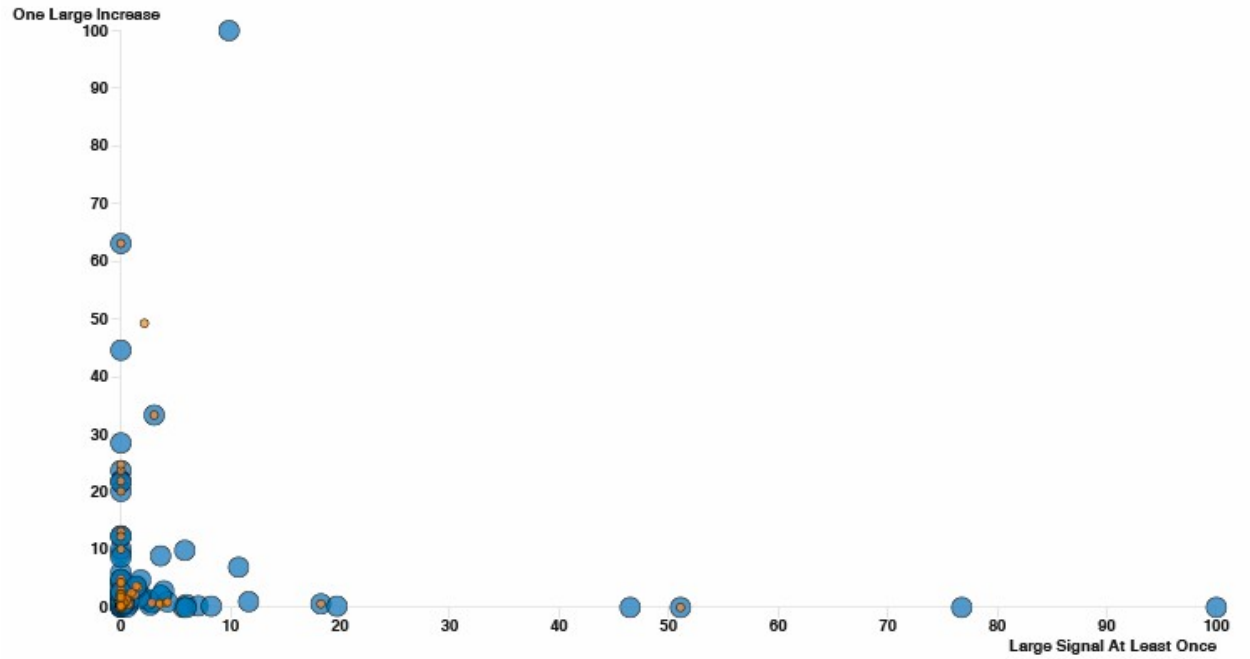


Рис. 3.3. Приклад одиночного графіка розсіювання. Поточне покоління показано світло-синім кольором; попереднє покоління - помаранчевим. (Знімок екрана з інструмента ISBST)

Користувач може вибрати одного або декількох кандидатів, що відображаються, і переглянути більш детальний вигляд. Цей вид включає, крім значень пристосованості щодо кожної мети, вхідні та вихідні значення, що показано на рисунку 3.5.

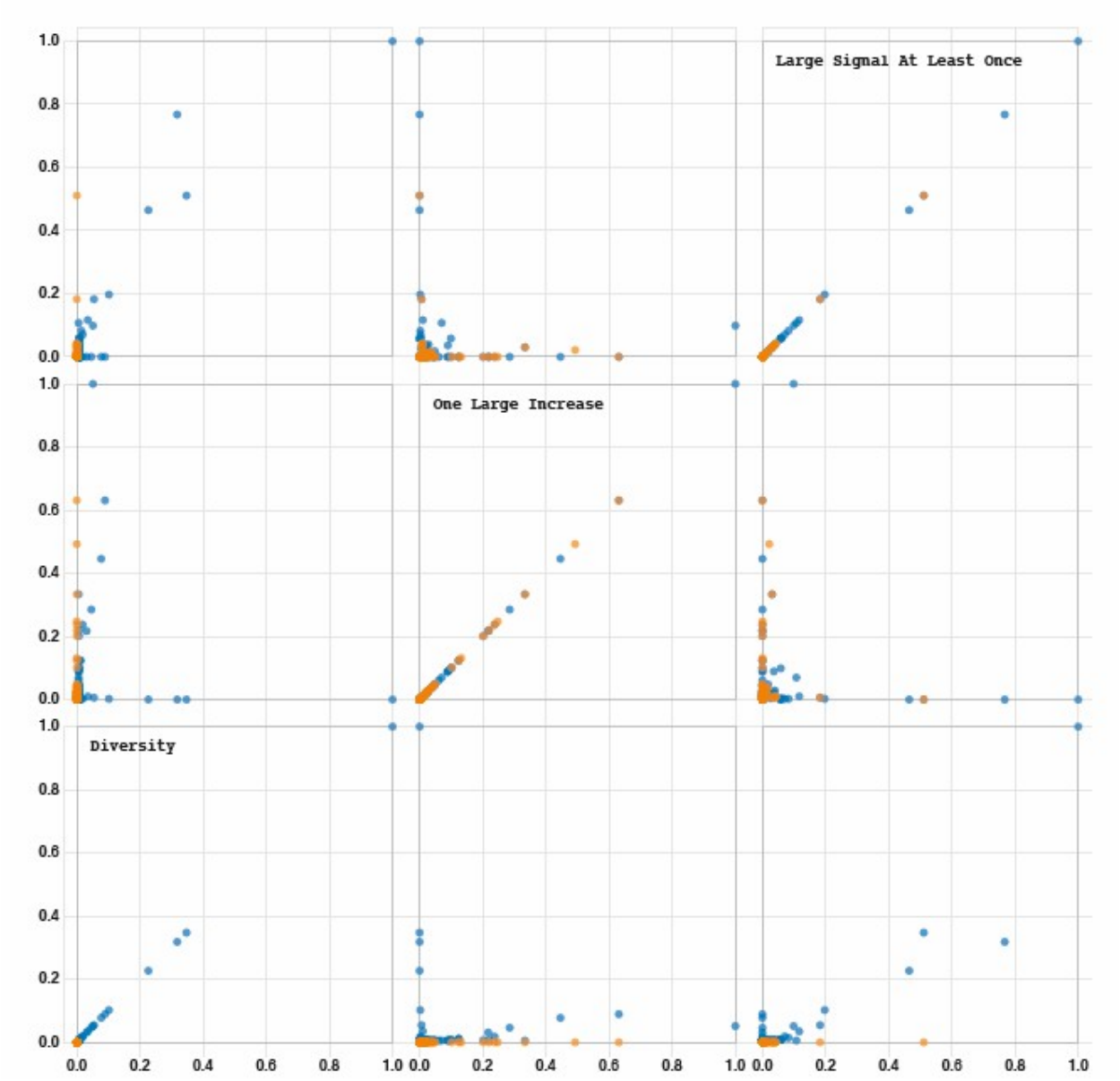
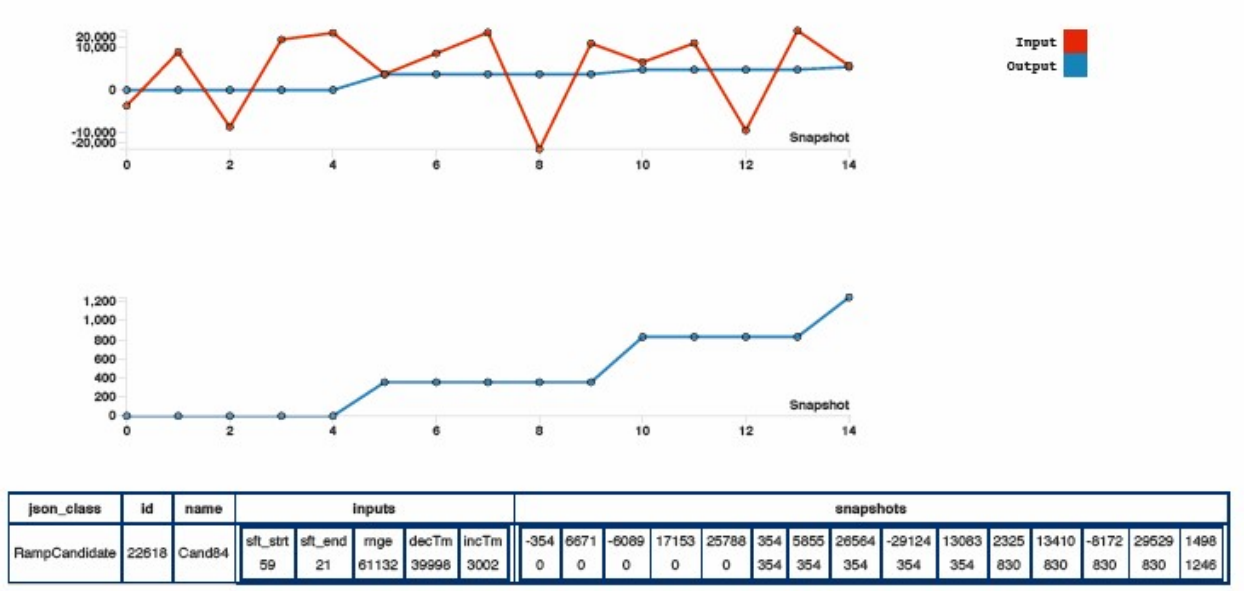


Рис. 3.4. Приклад матриці діаграми розсіювання. Поточне покоління показано світло-блакитним кольором; попереднє покоління помаранчевого кольору. (Знімок екрана з інструменту ISBST.)

Щоб керувати пошуком, фахівець галузі визначає відносну важливість цілей якості за допомогою панелі керування на рисунку 3.6. Відносна важливість виражається у вигляді ваг, всі з яких мають значення між weightmin = 0,0 і weightmax = 1,0, і модифікуються з кроком weight_delta = 0,1. Значення не повинні додаватися до будь-якого фіксованого значення. Ці ваги потім використовуються для обчислення IFF, як описано в підрозділі вище.

У будь-який момент користувач може експортувати тестові випадки, які, на його думку, є особливо цікавими для використання та дослідження поза системою ISBST.



json_class	id	name	inputs					snapshots													
			sft_strt	sft_end	rnge	decTm	incTm	-354	6671	-6089	17153	25788	354	5855	26564	-29124	13083	2325	13410	-8172	29529	1498
RampCandidate	22618	Cand84	59	21	61132	39998	3002	0	0	0	0	0	354	354	354	354	354	830	830	830	830	1246

Рис. 3.5. Детальний вигляд одного з кандидатів у тестовому прикладі. (Знімок екрану інструменту ISBST)

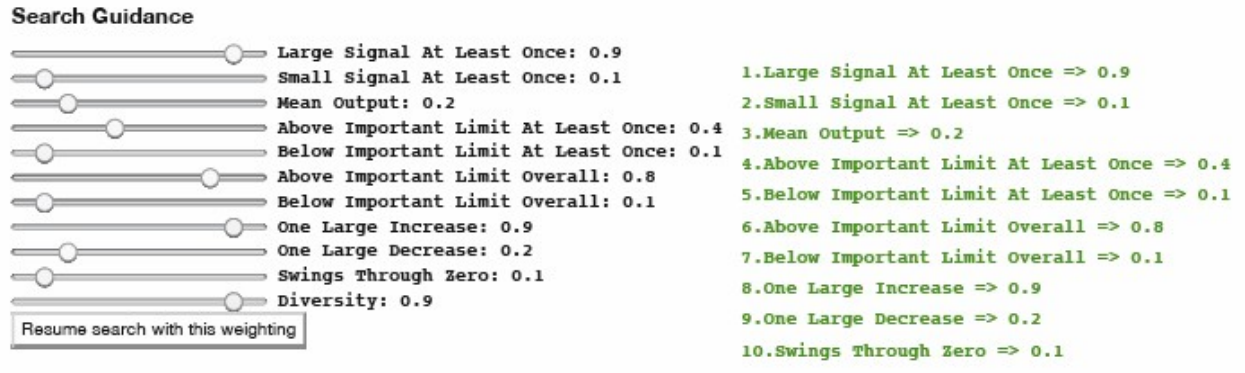


Рис. 3.6. Панель підказок пошуку. (Знімок екран інструменту ISBST)

3.3.4. Реалізація системи

Зовнішній цикл використовує комбінацію CoffeeScript, варіацію JavaScript, спрямовану на спрощення розробки, і бібліотеку Data-Driven Document (D3) для відображення кандидатів і отримання відгуків від фахівця галузі. D3 є бібліотекою JavaScript для маніпулювання та відображення даних

у браузері. Вона дозволяє динамічно створювати візуалізації, не впливаючи на зміну самих даних. Оскільки зовнішній цикл займається відображенням кандидатів, комбінація Coffeescript і бібліотеки D3 добре підходить для цієї мети.

Внутрішній цикл і інтерфейс SUT розроблені на Ruby. Це було обрано через відносну легкість взаємодії з іншими додатками, наприклад, SUT, інтерфейсами до симуляторів або інтерфейсами до апаратних стендів тестування, що дозволяє ISBST бути більш розширюваною в цілому.

Комунікація між внутрішнім і зовнішнім циклами здійснюється шляхом пакування об'єктів кандидатів у файл JavaScript Object Notation (JSON) і надання його через фреймворк Sinatra. Метаінформація, пов'язана з пошуком, наприклад, значення пристосованості для кожної мети, включається в об'єкт кандидата і, таким чином, доступна у зовнішньому циклі. Це дозволяє системі легше адаптуватися до змін у вимогах до відображення.

3.4. Методологія оцінки системи

Мета оцінювання полягала в тому, щоб спостерігати за тим, як інженери взаємодіють із системою ISBST, і визначити, чи можна успішно використовувати систему для розробки відповідних тестів. Учасники розробляли програмне забезпечення за допомогою DomainDevEnvironment або працювали над розробкою самого інструменту. Як наслідок, їх усіх можна вважати фахівцями в домені завдяки їхній поточній роботі з інструментами, специфічними для домену, а також їхньому попередньому досвіду діяльності в домені.

Подальші етапи оцінки були такими:

1. Збір інформації. Початковий набір із п'яти запитань використовувався для визначення досвіду і з SoftRamp SUT. Для тих, хто

брав участь у попередніх спробах перевірки, також було зібрано враження про них, оскільки вони могли вплинути на поточну оцінку.

2. Розуміння SBST і прототипу ISBST. Цей розділ містив коротку демонстрацію системи ISBST, її роботи, а також роз'яснення будь-яких практичних питань щодо використання системи ISBST.

3. Поточні процедури тестування. На цьому етапі кандидат обговорив поточні стратегії розробки тестів і встановив базову лінію для типу, кількості та якості тестів, які розробник може створити вручну (тобто без допомоги системи ISBST).

4. Практична оцінка. Під час цього етапу використовується система ISBST для розробки тестів для компонента SoftRamp. На цьому етапі не було доступно жодних відповідей чи роз'яснень. Взаємодія між суб'єктом і системою на основі пошуку реєструвалася, включаючи ваги, призначені кожній меті для кожного кроку взаємодії. Подальші нотатки про перевагу певних стратегій чи цілей, а також будь-яка інша інформація, яка вважалася цікавою, також була записана дослідником.

5. Підведення підсумків. Ця фаза складалася з короткого інтерв'ю для збору відгуків про систему, вражень щодо взаємодії та результатів тестування, а також давала суб'єкту можливість надати будь-які додаткові коментарі чи пропозиції.

Ця оцінка мала дві основні цілі:

a) оцінити, чи достатній потік інформації між системою ISBST і фахівцем домену для того, щоб останній міг зробити обґрунтований вибір і успішно керувати системою;

b) оцінити, чи може система ISBST розробити потенційні рішення порівнянної якості з рішеннями, виготовленими вручну.

Інтерфейс отримав переважно позитивну оцінку щодо першої цілі, згаданої вище. Хоча були виявлені деякі проблеми щодо чіткості подання доступної інформації, сама інформація була визнана корисною та цікавою.

Інформаційний потік був зрозумілим і корисним: учасники мали змогу спрямувати пошук.

На рисунку 3.7 показано огляд відповідей на фінальні запитання оцінювання після того, як кандидати мали можливість випробувати систему ISBST і використати її для розробки тестів.

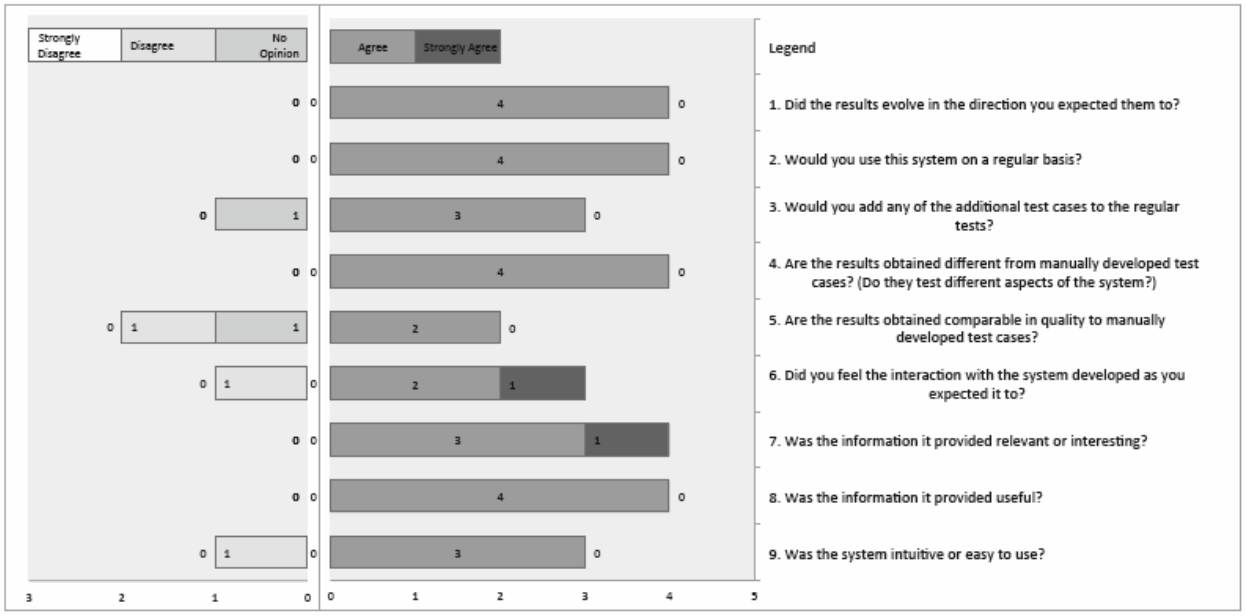


Рис. 3.7. Результати тестування

Перш за все, всі змогли без проблем або додаткової підтримки використовувати систему ISBST і розробляти тестові приклади для даного SUT. Інтерв’ю показало, що 3 з 4 учасників додадуть деякі тести, які вони розробили за допомогою інструменту ISBST, до стандартного набору тестів. Усі учасники погодилися з тим, що, навіть якщо тестові приклади кандидатів, які вони розробили, були не такими хорошими, як створені вручну, вони, схоже, досліджували різні та важливі області пошукового простору. Учасник 1 заявив, що один із знайдених ним тестів, включаючи значні коливання вхідного сигналу, був «тестером, на який людина-тестер може не подумати перевіряти». Це також показали інтерв'ю розбіжності в поглядах не є рідкістю навіть між розробниками, які працюють над подібними проектами, у схожому контексті та в одній компанії.

Загалом система ISBST дала хороші результати: лише після короткого навчання її використанню інженери змогли зрозуміти тестові випадки, які їм показали, обґрунтувати їх і успішно керувати пошуком.

3.4.1. Стратегії взаємодії з системою

Одним із визначних результатів оцінювання стала різноманітність стратегій, які використовували спеціалісти домену для взаємодії із системою. У цьому розділі будуть описані деякі різні підходи та їхні наслідки.

Обговорювані тут стратегії виникли з трьох основних джерел інформації. По-перше, початкове та заключне співбесіди містили запитання щодо підходу кожного учасника до тестування певного модуля. По-друге, уважно спостерігали за взаємодією учасників із системою, і спостереження збагатили початкове розуміння підходу учасників до тестування SUT. По-третє, були зібрані журнали подій взаємодії, щоб забезпечити кількісне уявлення про підхід учасника до використання системи ISBST. Всі учасники були поінформовані про тривалість етапу оцінювання, і можна припустити, що стратегія їх взаємодії була спрямована на найкраще використання наявного часу.

Перша помітна стратегія полягала в виділенні підмножини пов’язаних цілей якості на основі досвіду. Після першої події взаємодії учасник спробував зосередитися на кандидатах, які показали найкращі та найгірші загалом, і піддати їх більш ретельному аналізу. Під час обговорення учасники заявили, що їх поточний підхід до тестування полягає в дослідженні екстремальних значень. Їхній досвід показав, що більшість розломів виникла саме в цих областях. Було перевірено обмежений вибір тестових прикладів із проміжними значеннями для перевірки розумності. З точки зору взаємодії, ця стратегія призвела до обмеженої кількості подій взаємодії та невеликих варіацій у вагових коефіцієнтах, призначених різним обраним цілям якості. Це відбувається через те, що учасник зосереджує свою

увагу та час на цілях якості, намагаючись визначити цільові комбінації, які призвели до найбільших змін у тестових випадках.

Друга стратегія полягала в зосередженні на кандидатах, які сильно відрізнялися з точки зору однієї цілі якості, але дуже мало з точки зору інших. Пари таких кандидатів були ідентифіковані та відібрані, а детальні представлення кожного кандидата були використані для проведення порівняння. Взаємодія була зосереджена на аналізі графіків кожної пари кандидатів і спробі з'ясувати причину виявлених відмінностей.

Третя стратегія передбачала більше взаємодії з системою та розгляд кількох поколінь кандидатів. Зважування цілей якості значно змінювалося під час подій взаємодії, причому деякі цілі були виключені взагалі. Учасник зосередився на кандидатах, що виходять за межі, і шукав варіанти кандидатів, які демонстрували несподівані характеристики. Використання системи полягало в дослідженні груп вхідних даних, які зазвичай не включалися в тестові випадки вручну , і перевірці, чи виявляють будь-які викиди небажану поведінку. Взаємодія між цим учасником і системами була зосереджена на огляді всієї сукупності, виявленні викидів і розгляді відмінностей між поточним і попереднім поколіннями. Ця стратегія найбільше відповідала нашим очікуванням щодо використання системи.

На додаток до інформації, наданої різними стратегіями, підходи учасників до взаємодії підкреслювали мінливий характер особистого досвіду та очікувань. Важливо, щоб система, яка так сильно покладається на взаємодію та досвід її користувачів, допускала варіації з точки зору того, як ці користувачі взаємодіють з нею.

3.5. Імітаційний експеримент застосування системи

По-перше, було визначено чотири стратегії взаємодії на основі даних, отриманих з промислової оцінки. Найбільш складна взаємодія, спостережена в промисловій оцінці, була обрана для експерименту. Ця складна взаємодія є

реалістичною стратегією з незначними модифікаціями для кращого порівняння з іншими стратегіями. Було розроблено кілька спрощених версій реалістичної стратегії, щоб представити діапазон різних поведінок, спостережених під час оцінки. Ці спрощені стратегії можна детальніше побачити в таблиці 3.2.

Таблиця 3.2.

Огляд стратегій взаємодії, використаних для експерименту

No.	Strategy	Description	Comments
1	Null Strategy	All the objectives have the same weight: 0.5.	It is the default strategy of the system and the control group of the experiment.
2	Clear Fitness Function	The 5 selected objectives have weight 1, the rest 0.1.	It simulates a case where the domain specialist knows from the first step what type of fitness function they are looking for. While not realistic, it is a plausible case and a useful comparison.
3	Slowly Finding a Fitness Function	The weights of the objectives change, but the change is gradual. All the 5 selected objectives start at weight 0.1 and slowly increase in priority until reaching 1.	This simulates a case where the domain specialist starts from a different weighting, and slowly adjusts it until finding the goal. Though not completely realistic, such behavior has been observed during the industrial evaluation.
4	Realistic	Each of the 5 selected objectives is the top priority during two interaction steps, the overall sums of their weights are equal, and no two objectives are top priority simultaneously.	This captures the most realistic interaction, while preserving equal overall weights and allowing a comparison with the other strategies.

Реалістична стратегія є більш складною і ближчою до реальної поведінки фахівця галузі, тоді як нульова стратегія імітує поведінку системи пошукового тестування програмного забезпечення без взаємодії.

Реалістична стратегія зосереджена на оптимізації підмножини з 5 цілей якості. Вибір цілей був зроблений на основі журналів взаємодії, причому обрані цілі виявляли як синергію, так і протиріччя. Щоб забезпечити справедливе порівняння, всі спрощені версії використовують той самий підмножину цілей.

По-друге, ці стратегії були порівняні шляхом запуску системи ISBST з кожною стратегією та вимірювання загального значення пристосованості популяції.

Щоб забезпечити справедливе порівняння, для кожної стратегії було застосовано однакові «зусилля», і це було виміряно з точки зору оцінок пристосованості, зроблених алгоритмом пошуку. Усі стратегії мають однакову тривалість: 11 кроків взаємодії, що становить 3 300 оцінок пристосованості. Більше того, там, де стратегії вимагають зміни функції пристосованості шляхом переважування, загальна сума ваг є однаковою, щоб запобігти отриманню однією зі стратегій несправедливої переваги.

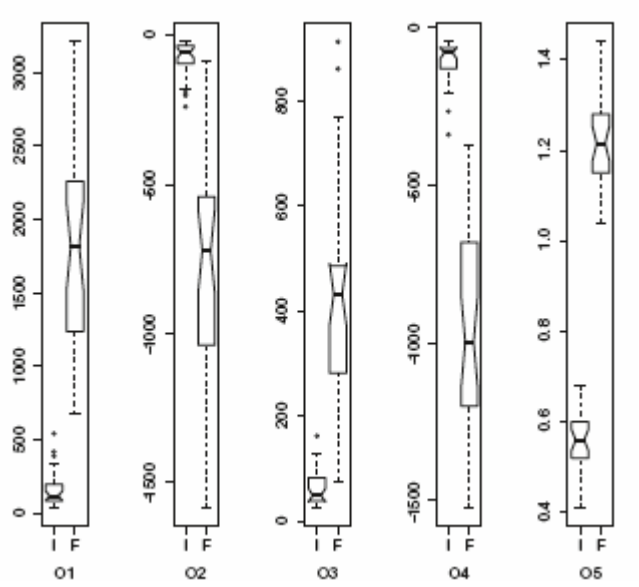
Система ISBST використовує початкову випадкову популяцію і застосовує стандартне зважування в проміжній функції пристосованості для створення популяції, яку користувач бачить на першому заході взаємодії. Тому порівняння проводиться між початковими популяціями та тими, що в кінці 11 кроків взаємодії.

Алгоритм пошуку є стохастичним, і тому, щоб мінімізувати вплив цієї випадковості на оцінку, кожна стратегія запускається загалом 30 разів. Для полегшення цього повторення стратегії автоматично застосовувалися скриптом, який замінив людину у зовнішньому циклі.

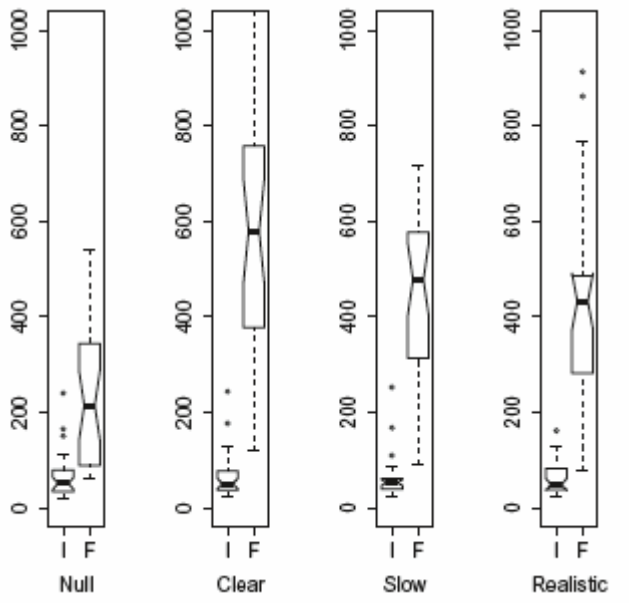
3.5.1. Представлення результатів

Перший аналіз порівняв початкову популяцію з кінцевою, використовуючи кожну з стратегій. Статистична значущість між початковою та кінцевою популяціями оцінювалася за допомогою парного тесту рангового суму Вількоксона над вибіркою з 30 результатів для кожної комбінації мети та стратегії. Для реалістичної стратегії для цілей 1-5 значення p були $p < 1e-06$. Порівняння початкових і кінцевих популяцій дає схожі значення для всіх стратегій.

Існувала значна різниця між усіма значеннями пристосованості цілей для всіх стратегій. Це чітко показує, що алгоритм пошуку, що лежить в основі внутрішнього циклу, явно впливає на результат, незалежно від обраної стратегії взаємодії.



а) Порівняння всіх цілей якості для реалістичної стратегії



б) Порівняння цілі якості o3 для всіх стратегій взаємодії. Об’єкт o3 було вибрано випадковим чином для відображення, але всі цілі якості демонструють однаковий ефект

Рис. 3.8. Детальний перегляд деяких експериментальних результатів.

Діаграми розмаху на лівій стороні рисунка 3.8 порівнюють зміну кожної з п’яти цілей якості при застосуванні реалістичної стратегії. Малюнок показує чітку різницю між початковою та кінцевою популяціями, тим самим

вказуючи на те, що базовий алгоритм пошуку працює так, як ми припускали. Метою для цілей 1, 3 і 5 було максимізувати значення пристосованості, тоді як значення цілей 2 і 4 мали бути мінімізовані, і результати на рисунку 3.8 відображають ці цілі.

Цікавим є порівняння, показане на правій стороні рисунка 3.8, що показує зміну значення однієї мети якості (o3) для всіх стратегій взаємодії. Відображена мета була обрана випадковим чином, але всі показують однакову поведінку для різних стратегій: Нульова стратегія працює набагато гірше за всіх, тоді як інші три стратегії набагато ближчі за значеннями пристосованості.

Порівняння між трьома рештою стратегіями, тобто Clear, Slow і Realistic, досить прямолінійне. Стратегія Clear показує функцію пристосованості, яка ніколи не змінюється, даючи найкращі результати, але припускаючи точне знання. Реалістична стратегія показує трохи гірші результати, але починає з найбільш реалістичного припущення: що фахівець галузі не може точно передбачити бажану функцію пристосованості з початку тесту. Крім того, був проведений парний тест рангового суму Вількоксона між кінцевою популяцією реалістичної стратегії та нульовою стратегією. Значення p для цього тесту для цілей 1-5 відповідно були $p < :001$, за винятком останньої мети, де $p = 0:6324$. Порівняння не є значущим щодо останньої мети, кількості разів, коли вихідний сигнал перетинає нуль. Через обмеження поточної реалізації значення, отримані для цієї мети, коливалися між невеликим набором значень. Для всіх інших цілей інтерактивна стратегія, навіть субоптимальна, явно перевершувала неінтерактивну нульову стратегію.

Наше припущення полягало в тому, що реалістична стратегія створить більш різноманітний набір рішень, але також матиме значно нижче загальне значення пристосованості. Результати цього експерименту, здається, вказують на те, що він працює краще, ніж очікувалося. Вони також, здається, показують, що вибір стратегії взаємодії не є критичним для отримання

хороших результатів. Використання стратегії Clear або Slow було б кращим, але якщо доступна інформація не дозволяє такого вибору, реалістична стратегія все одно забезпечить хороші результати.

Будь-яке обговорення результатів має починатися з твердження, що це було прикладне дослідження системи ISBST, розробленої для певного контексту та оціненої в одній конкретній компанії. Випадкове дослідження складалося з промислової оцінки та лабораторного експерименту, встановленого на основі інформації, отриманої в результаті оцінки. У результаті можна лише стверджувати, що ці висновки застосовні в цьому контексті, і на даному етапі неможливо зробити більш загальні висновки.

Тим не менш, ми вважаємо, що результати вказують на те, що ISBST є потенційно життєздатним інструментом для використання в промисловості. Фахівці домену нашого промислового партнера змогли після короткого навчального сеансу та за обмежений час створити цікаві тестові приклади. У деяких випадках створені тести були порівнянні за якістю з тестами, створеними вручну, і всі учасники погодилися, що тести були корисними з точки зору дослідження поведінки, яка наразі не охоплена тестами, або надали уявлення про роботу тестованих (SoftRamp) модуль.

Поточне дослідження допомогло показати деякі складності застосування системи ISBST у промисловому контексті. Доменні спеціалісти мають різний досвід і по-різному підходять до виконання своїх обов’язків, тоді як цілі пошуку можуть відрізнятися залежно від контексту. Тим не менш, методи, засновані на пошуку, можуть бути застосовані в промисловому контексті і наразі дали цікаві результати.

Висновки до розділу

В даному розділі розглянута техніка та методологія оцінки інтерактивного тестування програмного забезпечення, що є важливою складовою процесу розробки та вдосконалення програмних систем.

По-перше, було акцентовано увагу на показниках якості програмного забезпечення, які формують основу для оцінки будь-якої системи. Ці показники є ключовими для визначення ефективності, надійності та відповідності розробленого продукту очікуванням користувачів.

По-друге, проаналізовано існуючі підходи до інтерактивного еволюційного пошуку, що дають змогу забезпечити високий рівень адаптивності та точності при тестуванні складних архітектур програмного забезпечення. Було визначено переваги і недоліки різних методик, а також обґрунтовано потребу у вдосконаленні цих підходів.

По-третє, запропоновано концепцію побудови системи інтерактивного тестування програмної архітектури. Ця концепція включає детальний опис внутрішнього циклу роботи системи, алгоритму пошуку, зовнішнього циклу та способів реалізації. Внутрішній цикл забезпечує оптимізацію процесу тестування на основі зворотного зв’язку, а алгоритм пошуку спрямований на ефективне виявлення помилок.

По-четверте, була розроблена методологія оцінки системи, яка враховує різні стратегії взаємодії з нею. Цей підхід дозволяє комплексно оцінити функціональність, продуктивність і зручність використання системи з боку кінцевих користувачів.

По-п'яте, проведено імітаційний експеримент застосування системи, результати якого продемонстрували її ефективність у реальних умовах. Представлені результати дали змогу підтвердити правильність запропонованої концепції та доцільність її практичного використання.

Таким чином, результати досліджень цього розділу сприяють подальшому розвитку інтерактивного тестування програмного забезпечення. Запропонована методологія та концепція системи можуть бути використані для підвищення якості сучасних програмних продуктів.

ВИСНОВКИ

У магістерській роботі було проведено комплексне дослідження методів, методологій і підходів до інтерактивного тестування програмного забезпечення, спрямованого на підвищення якості та ефективності процесу розробки сучасних програмних систем.

Перший розділ присвячено теоретичному обґрунтуванню процесів тестування програмного забезпечення. Було розглянуто основні відомості про тестування на основі пошуку, визначено його ключові етапи, переваги та недоліки. Запропоновано концепцію інтерактивної системи тестування, яка враховує потреби користувачів у швидкості та точності виявлення дефектів. Крім того, сформульовано основні концепції системи тестування, що базується на еволюційному підході, як основи для її подальшої розробки.

Другий розділ зосереджено на розробці методів інтерактивного еволюційного пошуку для тестування програмного забезпечення. Було проаналізовано метаевристичні методи, що сприяють оптимізації пошуку рішень у складних і динамічних середовищах. Розглянуто існуючі підходи до еволюційного пошуку, а також розроблено методику застосування запропонованої системи на практиці. Особливу увагу приділено побудові прототипу системи, що дозволяє оцінити її функціональність і ефективність у реальних умовах. Проведено емпіричну оцінку прототипу, результати якої підтвердили його здатність виявляти дефекти з високим рівнем точності.

Третій розділ містить опис техніки та методології оцінки інтерактивного тестування. Було висвітлено важливість показників якості програмного забезпечення для об'єктивної оцінки системи, а також розглянуто сучасні підходи до інтерактивного пошуку. Запропоновано концепцію побудови системи інтерактивного тестування програмної архітектури, що включає внутрішній і зовнішній цикли роботи, алгоритм пошуку та стратегії взаємодії з користувачем. Проведено імітаційний експеримент, результати якого засвідчили ефективність розробленої системи

у виявленні дефектів програмного забезпечення та її доцільність у реальному використанні.

Основний науковий результат роботи полягає у розробці інтерактивної системи тестування програмного забезпечення, яка базується на еволюційних методах пошуку, та її успішній апробації в рамках імітаційного експерименту. Запропоновані методики та технічні рішення можуть бути використані для підвищення ефективності та якості програмного забезпечення, а також слугувати основою для подальших досліджень у цій сфері.

Результати роботи підтверджують актуальність і доцільність інтерактивного підходу до тестування, сприяючи розвитку інноваційних рішень у галузі програмної інженерії.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. P. M. Kruse, J. Wegener, and S. Wappler. \A Highly Configurable Test System for Evolutionary Black-box Testing of Embedded Systems". In: Proceedings of the 11th Annual Conference on Genetic and Evolution-ary Computation. GECCO '09. Montreal, Quebec, Canada: ACM, 2009, pp. 1545{1552. isbn: 978-1-60558-325-9.
2. C. Ebert and C. Jones. \Embedded Software: Facts, Figures, and Future". In: Computer 42.4 (2009), pp. 42{52. issn: 0018-9162.
3. B. Graaf, M. Lormans, and H. Toetenel. \Embedded software engineering: the state of the practice". In: IEEE Software 20.6 (2003), pp. 61{69. issn: 0740-7459.
4. P. Caspi, A. Sangiovanni-Vincentelli, L. Almeida, A. Benveniste, B. Bouyssounouse, G. Buttazzo, I. Crnkovic, W. Damm, J. Engblom, G.Folher, M. Garcia-Valls, H. Kopetz, Y. Lakhnech, F. Laroussinie, L.Lavagno, G. Lipari, F. Maraninchi, P. Peti, J. d. l. Puente, N. Scaife, J.Sifakis, R. de Simone, M. Torngren, P. Ver□ □ ssimo, A. J. Wellings, R. Wilhelm,T. Willemse, and W. Yi. \Guidelines for a Graduate Curriculum on Embedded Software and Systems". In: ACM Trans. Embed. Comput.Syst. 4.3 (Aug. 2005), pp. 587{611. issn: 1539-9087.
5. Sommerville, I. (2016). Software Engineering (10th ed.). Pearson.
6. Pressman, R. S., & Maxim, B. R. (2020). Software Engineering: A Practitioner’s Approach (9th ed.). McGraw-Hill Education.
7. Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). Wiley.
8. Ammann, P., & Offutt, J. (2016). Introduction to Software Testing (2nd ed.). Cambridge University Press.
9. Levenson, N. G. (1995). Safeware: System Safety and Computers. Addison-Wesley.

10. Koomen, T., & Pol, M. (1999). Test Process Improvement: A Practical Step-by-Step Guide to Structured Testing. Addison-Wesley.
11. Kaner, C., Falk, J., & Nguyen, H. Q. (1999). Testing Computer Software (2nd ed.). Wiley.
12. Black, R. (2009). Advanced Software Testing – Vol. 1: Guide to the ISTQB Certification as an Advanced Test Analyst (2nd ed.). Rocky Nook Computing.
13. Marick, B. (1995). The Craft of Software Testing: Subsystem Testing Includes (1st ed.). Prentice Hall.
14. Copeland, L. (2004). A Practitioner's Guide to Software Test Design. Artech House.
15. Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley.
16. Whittaker, J. A. (2003). How to Break Software: A Practical Guide to Testing. Addison-Wesley.
17. Jalote, P. (2008). Software Engineering: A Precise Approach. Wiley India.
18. Beizer, B. (1995). Black-Box Testing: Techniques for Functional Testing of Software and Systems. Wiley.
19. Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. Springer.
20. Fewster, M., & Graham, D. (1999). Software Test Automation: Effective Use of Test Execution Tools. Addison-Wesley.
21. Kloos, C. D., & Albert, L. (2018). Interactive Software Testing. Springer.
22. Liggesmeyer, P. (2009). Software Quality: Methods for Software and Systems Development. Springer.
23. Larman, C. (2004). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. Pearson.
24. He, H. (2009). Introduction to Automated Software Testing: Automated Testing Techniques. CreateSpace.
25. Paul, R. (2006). Software Testing: A Developer’s Perspective. Pearson.

26. Mathur, A. P. (2008). Foundations of Software Testing. Pearson.
27. Singh, Y. (2011). Software Testing. Cambridge University Press.
28. Weyuker, E. J. (1998). Software Testing Research and Practice. ACM SIGSOFT Software Engineering Notes.
29. Shroff, G. (2014). Agile Software Testing with Scrum and Kanban. Packt Publishing.
30. Daka, E., & Fraser, G. (2014). A Survey on Test Generation. ACM Transactions on Software Engineering.
31. Arcuri, A., & Briand, L. (2011). A Practical Guide to Testing Object-Oriented Software. Springer.
32. Marick, B. (1997). Functional Testing for Software Quality Assurance. Springer.
33. Pettichord, B. (2002). Test Design for Software Applications. IEEE Transactions on Software Engineering.
34. Lewis, W. E. (2005). Software Testing and Continuous Quality Improvement (2nd ed.). Auerbach Publications.
35. Fewster, M. (2001). Software Quality Assurance Testing for the Real World. Addison-Wesley.
36. Hazzan, O., & Dubinsky, Y. (2009). Agile Software Testing Methods. Springer.
37. K. Doganay, M. Bohlin, and O. Sellin. \Search Based Testing of Embedded Systems Implemented in IEC 61131-3: An Industrial Case Study". In: 2013 IEEE Sixth International Conference on Software Testing, Verication and Validation Workshops. 2013, pp. 425{432.
38. J. Wegener and M. Grochtmann. \Verifying Timing Constraints of Real-Time Systems by Means of Evolutionary Testing". In: Real-Time Systems 15.3 (1998), pp. 275{298.
39. Dustdar, S. (2010). Architectural Principles for Software Testing in Dynamic Environments. IEEE Transactions on Services Computing.

40. Ambler, S. (2002). The Agile Testing Revolution: How Agile Testing Changed Software. Wiley.
41. Kuhn, R. (2011). Systematic Software Testing Strategies. Springer
42. M. Ellims, J. Bridges, and D. C. Ince. \The Economics of Unit Testing". In: Empirical Softw. Engg. 11.1 (Mar. 2006), pp. 5{31. issn: 1382-3256.
43. P. Liggesmeyer and M. Trapp. \Trends in Embedded Software Engineering". In: IEEE Software 26.3 (2009), pp. 19{25. issn: 0740-7459.
44. M. Harman and B. F. Jones. \Search based software engineering". In: Information and Software Technology 43 (2001), pp. 833{839.
45. R. Feldt. \Genetic Programming as an Explorative Tool in Early Software Development Phases". In: Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering (SCASE '99). University of Limerick, Ireland: Limerick University Press, 1999, pp. 11{20.
46. M. Harman, S. A. Mansouri, and Y. Zhang. Search Based Software Engineering: A Comprehensive Analysis and Review of Trends Techniques and Applications. Tech. rep. TR-09-03. 2009.
47. G. Fraser and A. Arcuri. \1600 faults in 100 projects: automatically faults while achieving high coverage with EvoSuite". In: Empirical Software Engineering 20.3 (2015), pp. 611{639. issn: 1573-7616.
48. J. Campos, A. Arcuri, G. Fraser, and R. Abreu. \Continuous Test Generation: Enhancing Continuous Integration with Automated Test Generation". In: Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ASE '14. Vasteras, Sweden: ACM, 2014, pp. 55{66. isbn: 978-1-4503-3013-8.
49. M. Harman and P. McMinn. A theoretical and empirical study of search based testing: Local, global and hybrid search. IEEE Transactions on Software Engineering, 36(2):226{247, 2010.
50. A. J. Outt. Investigations of the software testing coupling e ect. ACM Trans. Softw. Eng. Methodol., 1:5{20, January 1992.

51. C. Pacheco and M. D. Ernst. Randoop:feedback-directed random testing for Java. In OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion, pages 815{816, New York, NY, USA, 2007. ACM.
52. N. Tillmann and J. N. de Halleux. Pex | white box test generation for .NET. In TAP 2008: International Conference on Tests And Proofs, volume 4966 of LNCS, pages 134 { 253. Springer, 2008.