УДК: 330.4 DOI 10.33111/mise.103.17

Труш М. С.,
кандидат економічних наук, доцент кафедри
системного аналізу та кібербезпеки,
КНЕУ імені Вадима Гетьмана
Колдов В. В.,
магістрант за освітньо-професійною програмою «Системний аналіз»,
КНЕУ імені Вадима Гетьмана

Trush M. S.,
PhD Candidate of Economic Sciences,
Associate Professor of the Department of System Analysis and Cybersecurity,
KNEU named after V. Hetman
Koldov V.V.,
Master's student of the educational and professional program
«System Analysis»,
KNEU named after V. Hetman

СПЕЦИФІКА ТЕСТУВАННЯ В ЕКОНОМІЧНИХ МОДЕЛЯХ: ТИПОВІ ПОМИЛКИ ТА РЕКОМЕНДАЦІЇ, ОГЛЯД ТЕХНІК ТЕСТ-ДИЗАЙНУ

SPECIFICITY OF TESTING IN ECONOMIC MODELS: TYPICAL ERRORS AND RECOMMENDATIONS, OVERVIEW OF TEST DESIGN TECHNIQUES

Анотація. *Економічне моделювання ґрунтується на вимогливій оцінці та валідації для точного прогнозування й розуміння економічних сценаріїв. Ця стаття детально розглядає методології тестування економічних моделей, наголошуючи на аналізі ризиків, перевірці прогнозів та оцінці впливу різноманітних економічних факторів. Вона підкреслює важливість тестування для точності відображення реальних економічних процесів та вдосконалення якості моделі.*

Висвітлено сім принципів тестування: наявність дефектів, неповне тестування, раннє тестування, кластеризація дефектів, парадокс пестицидів, контекстуальна залежність та помилкове уявлення щодо відсутності дефектів. Стандарти ISTQB класифікують тестування на функціональне, нефункціональне та тестування, пов'язане зі змінами. Стаття надає всебічну схему класифікації, визначає типові помилки під час тестування та пропонує рекомендації щодо їх уникнення.

Розглянуті типові помилки тестування, такі як припущення щодо можливості знаходження всіх дефектів. Рекомендації включають тестування з орієнтацією на цілі, акцент на негативному тестуванні та залучення кінцевих користувачів для ефективного уникнення помилок. Виділено значення тест-дизайну у створенні сценаріїв тестування та ефективному відстеженні охоплення вимог.

Ключові слова: *Тестування; типи тестування; принципи тестування; функціональне тестування; нефункціональне тестування; тестування, пов'язане зі змінами; ISTQB; класифікація тестування; типові помилки та рекомендації; техніки тест-дизайну.*

Abstract. *Economic modeling relies on rigorous assessment and validation to forecast and understand economic scenarios accurately. This paper delves into methodologies for testing economic models, emphasizing risk analysis, forecast validation, and evaluating diverse economic factors' impact. It underscores the importance of testing for precision in mirroring real economic processes and refining model quality.*

Seven principles of testing are elucidated: defect existence, incomplete testing, early testing, defect clustering, the pesticide paradox, contextual dependency, and the fallacy of error absence. ISTQB standards categorize testing into functional, non-functional, and changes-related testing. The paper provides a comprehensive classification scheme, identifies common testing errors, and offers recommendations to prevent them.

Typical testing errors, like assuming all defects can be found or solely attributing product quality to the testing team, are addressed. Recommendations include goal-oriented testing, emphasis on negative testing, and involving end-users to prevent errors effectively. The significance of test design in structuring test scenarios and tracking requirement coverage efficiently is highlighted.
Keywords: *Testing; types of testing; principles of testing; functional testing; non-functional testing; related to changes; ISTQB; testing classification; common mistakes and recommendations; test design techniques.*

Вступ. Економічне моделювання набуває особливої значущості для аналізу соціально-економічних процесів, що відбуваються в сучасному світі. У контексті структурування соціально-економічного простору від постіндустріального до інформаційного суспільства, верифікація й валідація моделей виявляються невід'ємною частиною розвитку суспільно-історичних шляхів та економічних сценаріїв. Важливою складовою досвіду в галузі моделювання інноваційних тенденцій економіки стає використання різноманітних методів і стратегій тестування економічних моделей, які дозволяють оцінити достовірність передбачень. Обговорення кращих практик та висвітлення результатів тестування сприяє покращенню якості економічних моделей і точності соціальних прогнозів.

Постановка проблеми у загальному вигляді. Варто зауважити, що проблеми у сфері економічного моделювання виникають з нестабільності економічних умов і труднощів передбачення поведінки ринків. Складність економічних взаємозв'язків і нестабільність ринків викликають потребу в надійних методиках прогнозування економічних змін, а також в аналізі їх впливів на ринкові процеси. Вирішення наведених проблем вимагає уваги до розробки нових підходів і впровадження високоефективних інструментів й методів тестування, які б допомагали вдосконалювати якість економічних моделей та підвищувати точність прогнозів.

Отже, проблеми розвитку адекватних економічних моделей для прогнозування економічних сценаріїв набувають **актуальності**, привертаючи до себе увагу як науковців-теоретиків, так і практиків у різних галузях знань.

Виділення невирішених раніше частин загальної проблеми. Виникнення зазначених проблем у сфері економічного моделювання викликає необхідність вирішення суттєвих питань, що стосуються вдосконалення методів тестування для підвищення ефективності, точності та надійності моделей. Значимим аспектом є оптимізація методів оцінки ризиків для прогнозування економічних моделей. Дотепер не вирішені питання щодо врахування впливу екзогенних факторів на моделі, що є важливим для підвищення їх точності в реальних умовах. Відсутність уваги до цих питань викликає необхідність обґрунтування й розробки практичних рекомендацій щодо вдосконалення методів і методик підвищення точності економічного моделювання, які можуть виявитися дієвими для розуміння розвитку соціально-економічних систем та подальшого прогнозування їх поведінки.

Аналіз останніх досліджень та публікацій показує, що даними проблемами активно займаються вчені як українського, так і міжнародного наукового співтовариства. Іноземні науковці Бем В., Бріджес Д., Еллімс М., Касурінен Д., Мартін Р., Салліван К., Хіннінен Т. проводять дослідження у напрямку вдосконалення методів тестування економічних моделей. Українські дослідники Дідковська М. В., Сьомкіна Т. В., Терентьєв О. О., Тимошенко Ю. О., Шабала Є. Є., Юрченко В. В. пропонують підходи, спрямовані на розвиток точних і надійних методів тестування економічних моделей, що виявляється важливим в контексті прогнозування економічних сценаріїв та їх впливу на ринкові процеси.

Касурінен Д. аналізує компоненти, важливі для тестування в організаціях, зосереджуючись на реальних програмних компаніях, виявленні ключових складових тестового процесу та спостереженні за роботою з тестування на практиці для аналізу можливостей удосконалення організаційного тестування. Дідковська М.В. відзначає актуальність якісного програмного забезпечення з економічної перспективи, наголошуючи на стабільності рівня помилок у програмному коді протягом останніх 20 років, незважаючи на застосування новітніх технологій. Витрати через недоліки у програмному забезпеченні за даними Національного інституту по стандартах та технології США сягають мільярдів доларів щорічно, що становить близько 1% національного валового внутрішнього продукту країни.

Метою статті є проведення аналізу та обґрунтування новітніх підходів у тестуванні економічних моделей, визначення їх ефективності та впливу на точність прогнозів соціально-економічних сценаріїв.

Виклад основного матеріалу. Успішне моделювання економіки вимагає як створення, так і ретельного тестування моделей на точність відображення реальних економічних процесів. Розглядаючи різні аспекти моделювання економіки, ми акцентуємо увагу на методах тестування економічних моделей: аналізі ризиків, валідації прогнозів, оцінках впливу різноманітних економічних чинників на результати моделювання. Адекватне тестування моделей ґрунтується на фундаментальних методологічних принципах, які необхідно розглянути для поглибленого розуміння теоретичних підходів і практичних методик використання тестового інструментарію. Наведемо специфікацію принципів тестування:

1. Тестування демонструє наявність дефектів: воно може показати, що дефекти присутні, але не може довести їх відсутність.

2. Повне, вичерпне тестування, з використанням всіх комбінацій вводів і передумов, недосяжно, фізично неможливо. Для більш точного фокусування зусиль з тестування потрібно використовувати аналіз ризиків і розстановку пріоритетів.

3. Раннє тестування: для якісного визначення дефектів тестування має розпочинатись якомога раніше в життєвому циклі розробки програмного забезпечення або системи, і повинно бути сфокусоване на певних цілях.

4. Скупчення (кластеризація) дефектів: зусилля тестування повинні бути зосереджені пропорційно очікуваній, а пізніше реальній щільності дефектів по модулях. Зазвичай, більшість виявлених під час тестування дефектів, які призводять до основної кількості системних збоїв, знаходяться в обмеженій кількості модулів. Тому, якщо знайдено баг, потрібно шукати ще в цьому ж місці.

5. Парадокс пестициду, що виникає при багаторазовому використанні одних й тих самих тестів — в кінцевому рахунку даний набір тестових сценаріїв перестає знаходити нові дефекти, всі тести проходять. Для подолання цього парадоксу тестові сценарії мають регулярно рецензуватись і коригуватись, а нові тести повинні бути різнобічними, щоб охопити всі компоненти програмного забезпечення, або системи, і знайти якомога більше дефектів.

6. Методи тестування варіюються в залежності від контексту. Тобто, безпеку програмного забезпечення перевіряють інакше, ніж у випадку інтернет-магазину.

7. Помилкове уявлення щодо відсутності дефектів: виявлені та виправлені дефекти не гарантують задоволення користувача, якщо система не відповідає його потребам та очікуванням.

У класифікаційній структурі тестування доцільно виділити різні функціональні складові: типи, види, рівні, напрямки.

Тестування класифікується за типами наступним чином.

Ручне тестування — процес, при якому людина виконує тести без застосування автоматизованих засобів. Тестувальник повинен мати певні особливості, такі як терплячість, уважність, креативність, вміння проводити нетрадиційні експерименти та розуміти внутрішні процеси системи, у тому числі роботу з базами даних.

Автоматизоване тестування — набір інструментів, що дозволяє автоматизувати завдання в процесі тестування. Спеціальні програмні засоби виконують частину чи всі тести; однак розробка тестів, підготовка даних, аналіз результатів тестування та створення звітів про виявлені проблеми залишаються відповідальністю людини [1].

Позитивне тестування спрямоване на дослідження додатку в ситуації строгого виконання дії за інструкцією, без будь-яких помилок, відхилень, введення невірних даних і т. ін. Якщо позитивні тест-кейси завершуються помилками, це тривожна ознака: додаток не працює належним чином навіть в ідеальних умовах (і можна припустити, що в неідеальних умовах він працюватимє ще гірше). Для прискорення тестування декілька позитивних тест-кейсів можна об’єднувати (наприклад, перед відправкою заповнити всі поля форми вірними значеннями); іноді це може ускладнити діагностику помилки, але суттєва економія часу компенсує ризик.

Негативне тестування перевіряє роботу програми в умовах некоректних операцій чи з використанням потенційно помилкових даних (наприклад, ділення на нуль). Оскільки в реальності таких ситуацій багато, негативні тест-кейси переважають за кількістю над позитивними. Проте, їх об’єднання може спричинити неправильне розуміння реакції програми та призвести до пропуску дефектів [2].

Прокласифікуємо тестування згідно зі стандартами ISTQB (рис. 1):

Рис. 2 Види тестування (Джерело: розроблено автором)

Розглянемо детальніше різні види тестування.

Тестування функціональності спрямоване на перевірку того, чи працює функціонал програми коректно, тобто, чи відповідає виконання функцій додатку вимогам, що до них пред'являються (рис. 2). В даному випадку тестується те, «що» система повинна робити. Таке тестування часто пов'язують із тестуванням за методом чорного ящика, проте метод білого ящика також може використовуватись для перевірки правильності реалізації функціональності [3].

Тестування безпеки орієнтоване на перевірку можливості програми уникати намагань несанкціонованого доступу до даних чи функцій.

Тестування взаємодії або сумісності спрямоване на перевірку того, як добре програма працює у визначеному середовищі; воно включає перевірку сумісності з апаратним забезпеченням, операційною системою та мережевою інфраструктурою (тестування сумісності конфігурацій) [3].

Рис. 3 Функціональні види тестування
(Джерело: розроблено автором)

Нефункціональне тестування перевіряє нефункціональні особливості програми: зручність використання, сумісність, продуктивність, безпеку (рис. 3). У цьому випадку перевіряється «як/яким чином» система виконує певні дії або функції.

Тестування продуктивності полягає у визначенні швидкості реагування програми на зовнішні впливи в умовах різного навантаження, враховуючи його характер та інтенсивність.

Тестування навантаження — процес перевірки можливостей програми зберігати визначені показники якості під час навантаження в межах прийнятних стандартів, а також при перевищенні цих меж для визначення запасу міцності системи.

Рис. 4. Нефункціональні види тестування (Джерело: розроблено автором)

Тестування надійності — процес перевірки можливості програми працювати в зазначених умовах певний період часу та/або виконувати певну кількість запитів.

Тестування стабільності — дослідження можливостей програми під час тривалого (протягом декількох годин) тестування з помірним рівнем навантаження.

Тестування на великі обсяги (об'ємне тестування) — аналіз продуктивності програми при обробці різних об'ємів даних, які, як правило, є великими.

Стрес-тестування досліджує, як додаток поводиться в ситуаціях, коли навантаження на нього значно перевищує очікуваний рівень або доступні ресурси для додатку обмежені. Це випробування може відбуватись навіть без контексту навантажувального тестування; у такому випадку воно представляє собою певну форму негативного тестування і називається «тестуванням на вибух».

Тестування зручності використання має на меті вивчення того, наскільки легко кінцевий користувач розуміє принципи роботи продукту і наскільки приємно йому користуватися ним. Успіх продукту часто залежить від емоційної реакції користувачів на нього. Для вдалого проведення цього тестування потрібні серйозні соціологічні та маркетингові дослідження з опитуванням кінцевих користувачів.

Тестування на відмовостійкість і відновлення — це процес, при якому емулюються або створюються критичні ситуації для перевірки можливості програми вчасно реагувати, правильно виявляти, ефективно вирішувати проблеми, що виникають у системах відновлення, з метою забезпечення безпеки, цілісності та продуктивності програми під час виникнення непередбачуваних ситуацій.

Функціональне та нефункціональне тестування спрямовані на виявлення дефектів у програмному продукті для їх виправлення, а також на оцінку відповідності програми вимогам та очікуванням клієнта. Це допомагає приймати рішення про якість продукту та готовність до передачі його клієнтові.

З'ясуємо характеристики тестування, яке пов'язане зі змінами (рис. 5).

Тестування збірки — набір автоматичних тестів, який перевіряє інтеграцію нової збірки, її стабільність і ключові функціональні можливості. Застосовується, коли на проєкті спостерігається висока частота випусків білдів. Тестування збірки запускається на кожній новій білді, перед випуском для подальшого тестування.

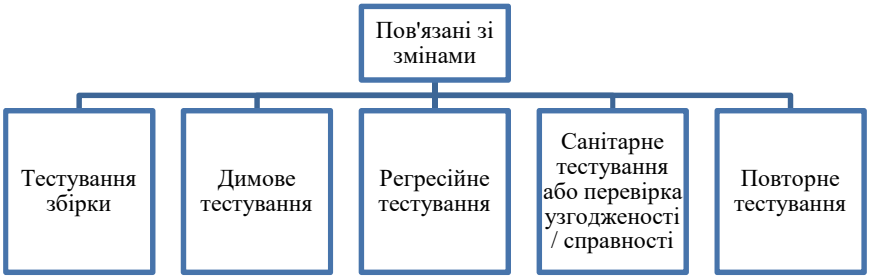

Рис. 5 Види тестування, пов’язані зі змінами
(Джерело: розроблено автором)

Регресійне тестування спрямоване на запобігання появи помилок в раніше працездатній функціональності, що може бути викликано змінами в додатку або середовищі його функціонування. Фредерік Брукс у своїй книзі «Міфічний людино-місяць» писав: «Фундаментальна проблема при супроводі програм полягає в тому, що виправлення однієї помилки з великою ймовірністю (20-50%) тягне появу нової» [4]. Тому регресійне тестування є невід’ємним інструментом забезпечення якості і активно використовується практично в будь-якому проєкті. Повторне тестування — перевірка тест-кейсів, що раніше виявили дефекти, з метою підтвердження усунення дефектів на фінальній стадії життєвого циклу, переведення дефекту в стан «перевірений» і «закритий» (звіт про дефект). Варто відзначити різницю між регресійним і повторним тестуванням. Повторне тестування перевіряє виправлення дефектів; регресійне — те, чи не вплинуло виправлення дефектів на інші модулі ПО та чи не викликало нових дефектів.

Димове тестування спрямоване на перевірку ключової функціональності, непрацездатність якої робить безглуздою саму ідею використання програми (або іншого об’єкта, що піддається димовому тестуванню). Санітарне тестування — вузько-спрямоване тестування, достатнє для доказу того, що конкретна функція працює згідно із заявленими в специфікації вимогами. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін, вироблених в ній або навколишньому середовищу. Зазвичай виконується вручну. Різниця між санітарним і димовим тестуванням полягає в глибині тестування функціональності.

ISTQB пропонує піраміду тестування, яку можна розділити на певні рівні:

Модульне тестування спрямоване на перевірку окремих невеликих частин програми (ізольовано від інших подібних частин). В даному випадку перевіряються окремі функції або методи класів, самі класи, взаємодія класів, невеликі бібліотеки, окремі частини програми. Зазначений вид тестування реалізують програмісти.

Інтеграційне тестування спрямоване на перевірку взаємодії між частинами додатка (кожна з яких перевірена окремо на стадії модульного тестування).

Системне тестування спрямоване на перевірку програми як єдиного цілого, складеного з частин, тестованих на модульній та інтеграційній стадіях.

Приймальне тестування (Acceptance Testing) — формалізоване; спрямоване на перевірку додатку кінцевим користувачем/замовником і винесення рішення щодо прийняття роботи у виконавця/проєктної команди; складається з користувацького, експлуатаційного, контрактного, правового; альфа та бета тестування.

Розглянувши специфічні характеристики різних видів тестування, проаналізувавши їх типологічні особливості та визначивши рівневу структуру, можемо скласти узагальнену схему класифікації тестування (рис. 6).

За наведеною схемою визначимо основні напрямки тестування.

Статичне тестування — тестування без запуску коду на виконання. В рамках цього підходу можуть потрапити під вплив: документи (вимоги, тест-кейси, описи архітектури додатку, схеми баз даних); графічні прототипи (наприклад, ескізи призначеного для користувача інтерфейсу); підготовлені тестові дані; параметри (настройки) середовища виконання програми; код додатка (часто виконується самими програмістами в рамках аудиту коду, що є специфічною варіацією взаємного перегляду в застосуванні до вихідного коду). Код додатка можна перевіряти і з використанням технік тестування на основі структур коду.

Динамічне тестування — тестування з запуском коду на виконання: програми вцілому/системне тестування, декількох взаємопов'язаних частин/інтеграційне тестування, окремих частин/модульне чи компонентне тестування, окремих ділянок коду. Основна ідея полягає в перевірці реальної поведінки (частини) додатку.

На завершення аналізу класифікаційних параметрів тестування варто визначити наступні типові помилки при виконанні процедур тестування, такі як переконання у можливості знайти всі дефекти, віднесення відповідальності за якість лише до тестувальників, обмежене бачення мети тестування лише пошуком дефектів.

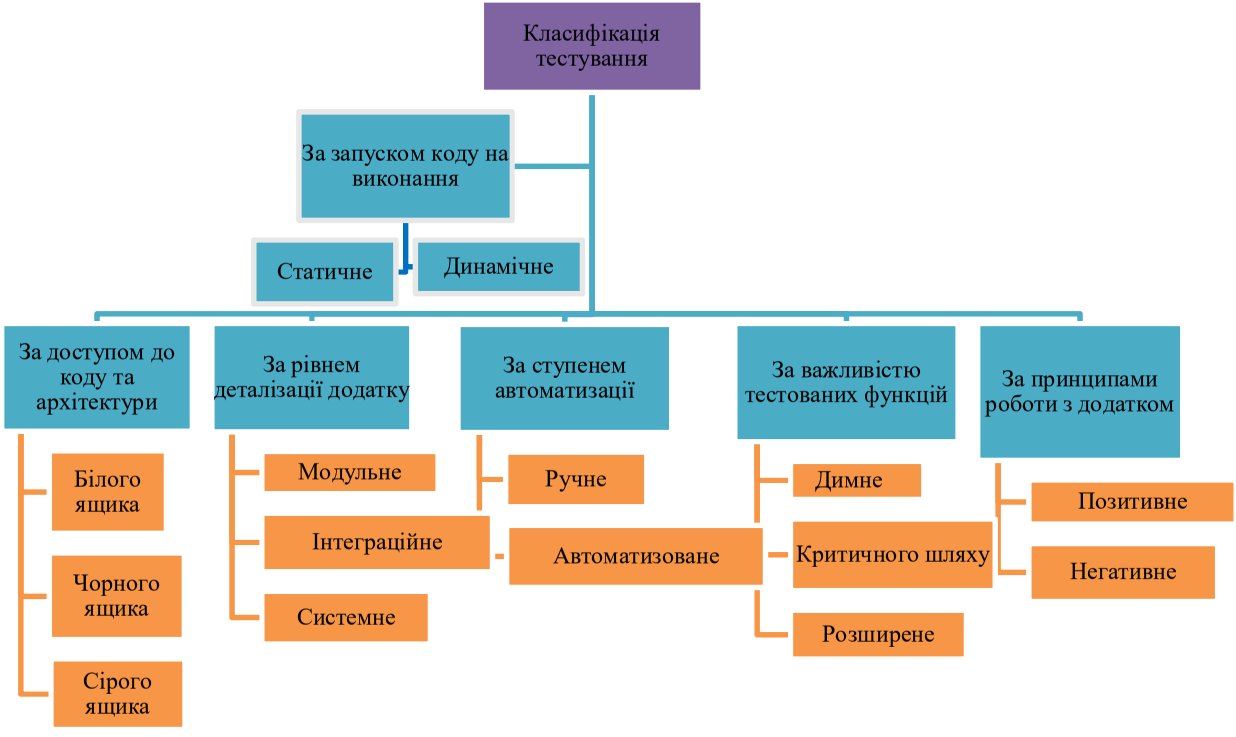

Рис. 6 Класифікація тестування (Джерело: розроблено автором)

Усвідомивши ці помилки, можна запропонувати загальні рекомендації для їх уникнення: важливо починати тестування з чітко визначених цілей; визнавати тест-кейси, які виявляють помилки; акцентувати увагу на негативному тестуванні; знати, коли завершити тестування; уникати перевірки власної роботи; залучати кінцевих користувачів; зосереджуватись на найбільш затребуваних функціях.

Класифікація виділених параметрів тестування дозволяє визначити підходи та методи, які можна використовувати для перевірки економічних моделей.

Метод чорного ящика — підхід у тестуванні, при якому тестувальник обмежений у доступі до внутрішньої структури чи коду програми (або свідомо не звертається до них), а також не має достатньо знань для їх аналізу. Більшість видів тестування користуються цим методом. Тестувальник здійснює вплив на програму так, як це роблять користувачі або інші додатки у реальних умовах експлуатації [5].

Метод білого ящика передбачає доступ тестувальника до внутрішньої структури та коду додатка, а також достатньо знань для їх розуміння. Іноді метод пов'язують зі статичним тестуванням: проте, ніщо не заважає запустити код на виконання і при цьому періодично звертатись до самого коду; а модульне тестування і взагалі передбачає запуск коду на виконання і роботу з ним, а не з додатком повністю [6].

Комбінацією вищезазначених методів є метод сірого ящика, при якому тестувальник має доступ лише до частини коду і архітектури, тобто внутрішня структура тестованого об'єкта відома частково і з'ясовується у ході дослідження. Явна згадка методу — рідкісний випадок: зазвичай говорять про методи білого або чорного ящика в застосуванні до певних частин додатку, при цьому розуміючи, що додаток повністю тестується за методом сірого ящика. Можна зустріти визначення методу сірого ящика як протиставлення методам білого і чорного ящика.

Близьким до методу білого ящика виступає тестування на основі тест-дизайну. Тест-дизайн — етап створення тестових сценаріїв та організації тестування для проєкту, що має на меті — структурувати процедури контролю якості, забезпечуючи ефективне покриття вимог та мінімізацію кількості тестів для перевірки продукту. Використання конкретних технік дизайну залежить від контексту, об'єму, рівня, типу тестування. Для модульного тестування можна використовувати методи критичних шляхів або рівних класів з метою переконатись в адекватнос-

ті окремих компонентів програми. На вищих рівнях інтеграційного чи системного тестування, застосування методів тестування перехресних залежностей чи граничного значення може допомогти перевірити взаємодію між компонентами та системою вцілому [7].

Розглянемо та охарактеризуємо найбільш популярні техніки тест-дизайну.

Перехід станів візуалізує стани програмної системи на різних часових інтервалах та етапах використання. Візуальну інформацію легше сприймати у порівнянні з вербальним описом. Тому перехід станів дозволяє швидше прийти до кінцевого тестового покриття. Дана техніка ефективна для створення наборів тестів для систем, які мають багато варіацій станів; вона буде корисною, якщо тестується послідовність подій з обмеженою кількістю варіантів вхідних даних.

Розбиття даних на еквівалентні класи ефективне для великих об'ємів вхідних даних чи великої кількості однакових вхідних варіацій, якщо компоненти схожі і можуть бути об'єднані в загальну групу. В даному випадку тестується лише кілька значень з кожної групи; це не гарантує, що решта значень будуть без помилок. Використання декількох елементів з групи може виявитись досить ілюстративним.

Аналіз граничних значень базується на еквівалентному розбитті на класи. Однак, групування даних в еквівалентні класи тут відбувається за відсутності перевірки значень з певного класу. Перевіряються граничні значення, що знаходяться на кордонах класів. Ця логіка чудово працює і для інтеграційного тестування. Під час модульного тестування ми перевіряємо менші елементи, а на наступному рівні помилки, швидше за все, з'являться на стиках модулів.

Розглянемо *модельну задачу 1: Пошук граничних значень.* Тестування сайту електронної комерції. При замовленні можна вказати кількість товару. Під цим полем є напис «Акційна ціна для оптового замовлення». Проте, не вказано: яка кількість для цього товару вважається оптовим замовленням; кількість наявного товару на складі. Але відомі мінімальне і максимальне значення для самого поля.

Вирішення. Для вирішення задачі ми скористалися технікою тест-дизайну, а саме використали аналіз граничних значень. Для більшої оптимізації було написано код на мові програмування Python, який дозволяє дізнатись граничні значення, виходячи з таких вхідних даних — мінімальне та максимальне значення поля.

Прог. 1. Автоматизоване рішення для пошуку граничних значень

```
  import random
  def generate_second_array(start_range, end_range, length):
    if (end_range – start_range) >= length:
     random_value = random.randint(start_range + length, end_range)
     start_value = random_value – length
     end_value = start_value + length

     second_array = list(range(start_value, end_value))
     return second_array
    else:
     print(«Довжина другого масиву перевищує довжину першого»)
     return []

  def find_common_elements(first_array, second_array):
    common_elements = []

    for value in second_array:
     start = 0
     end = len(first_array) – 1

     while start <= end:
      mid = (start + end) // 2

      if first_array[mid] == value:
       common_elements.append(value)
       break
      elif first_array[mid] < value:
       start = mid + 1
      else:
       end = mid – 1
    return common_elements

  # Введення користувача для створення першого масиву
```

```
  start_range = int(input(«Введіть мінімальне значення для поля: «))
  end_range = int(input(«Введіть максимальне значення для поля: «))
  first_array = list(range(start_range, end_range))

  # Генерація другого масиву
  second_array_length = random.randint(start_range, end_range)
  second_array = generate_second_array(start_range, end_range, second_array_length)

  # Пошук елементів другого масиву
  common_elements = find_common_elements(first_array, second_array)
  print(«Мінімальна кількість для оптової ціни:», common_elements[1])
  print(«Максимальна кількість у наявності:», common_elements[-1])
```

Пояснення до коду: Цей код крок за кроком реалізує автоматизоване рішення для пошуку граничних значень у масивах:

- generate_second_array() — функція генерує другий масив чисел вказаної довжини у визначеному діапазоні; якщо довжина другого масиву не перевищує довжину першого, то вона генерує випадкове число у вказаному діапазоні та створює масив чисел відповідно до цього числа.
- find_common_elements() — функція приймає два масиви та знаходить спільні елементи між ними, використовуючи для цього бінарний пошук у першому масиві для кожного елемента другого масиву.
- Користувач вводить мінімальне та максимальне значення для створення першого масиву.
- Створюється перший масив чисел від мінімального до максимального значення.
- Генерується випадкова довжина для другого масиву у вказаному діапазоні.
- Генерується другий масив чисел з урахуванням випадкової довжини, викликаючи функцію generate_second_array().
- Пошук спільних елементів між першим та другим масивами за допомогою функції find_common_elements().
- Вивід у консоль мінімального значення для оптової ціни (індекс 1 у відсортованому масиві спільних елементів) та максима-

льної кількості у наявності (останній елемент у відсортованому масиві спільних елементів).

Приклад застосування: Мінімальне значення поля — 0. Максимальне значення поля — 9999. Треба знайти: Мінімальну кількість товарів, при якому діє оптова знижка та залишок товарів на складі.

Результат: Введіть мінімальне значення для поля: 0**.** Введіть максимальне значення для поля: 9999**.** Мінімальна кількість для оптової ціни: 658**.** Максимальна кількість у наявності: 8833.

Висновки. У ході дослідження, яке було проведене для написання даної статті, було розглянуто принципи, типи, види, рівні, методи і техніки тест-дизайну. Для техніки тест-дизайну «Аналіз граничних значень» було розроблено код, який дозволяє знаходити граничні значення на певному проміжку.

Класифікація виділених параметрів тестування дозволяє підкреслити різноманітність підходів і методів, які можна використовувати для перевірки різних аспектів побудови економічних моделей. Типові помилки тестування та універсальні рекомендації надають контекст для розуміння, на що слід звертати увагу при проведенні тестування та як уникнути поширених помилок.

Таким чином, застосування цих принципів і методів може сприяти підвищенню надійності та точності економічних моделей, спрямованих на прогнозування та аналіз розвитку економіки.

Бібліографічні посилання

1. Kaner C. Testing Computer Software / C. Kaner, J. Falk, H. Q. Nguyen — 2nd Edition. — Hoboken: Wiley, 1999. — 324 c.
2. Whittaker J. How to Break Software: A Practical Guide to Testing / J. Whittaker — New York:Pearson, 2002. — 74 c.
3. ISTQB (International Software Testing Qualifications Board) https://www.istqb.org/ — (дата звернення 10.11.2023).
4. Frederick Phillips Brooks, Jr. The Mythical Man-Month: Essays on Software Engineering (Фредерік П. Брукс, Міфічний людино-місяць: Есе з програмної інженерії). — Massachusetts, Reading, Addison-Wesley Professional, 1995. — 322 pg.
5. Weinberg G. M. Perfect Software And Other Illusions about Testing / G. M. Weinberg [S. l.] : Dorset House, 2011. — 200 c.
6. Myers G. J. The Art of Software Testing / G. J. Myers, C. Sandler, T. Badgett — Hoboken: Wiley, 2011. — 173 c.
7. Ministry of Testing https://www.ministryoftesting.com/ — (дата звернення 10.11.2023).

Статтю подано до редакції: 27.11.2023