DOI 10.31392/UDU-nc.series2.2024.23(30).04
УДК 378:004.05

Закусило Микола Миколайович,
аспірант кафедри інформаційних систем і технологій
Українського державного університету імені Михайла Драгоманова,
вул. Пирогова 9, Київ, Україна
ORCID ID 0009-0004-9355-0980
nikola.zakusilo20071991@gmail.com

Шевчук Борис Вікторович,
кандидат педагогічних наук, доцент,
доцент кафедри інформаційних систем і технологій,
Українського державного університету імені Михайла Драгоманова,
вул. Пирогова 9, Київ, Україна
ORCID ID 0000-0002-4406-1011
sh.bera04@gmail.com

ОГЛЯД ТЕХНІК ТЕСТ ДИЗАЙНУ В КОНТЕКСТІ НАВЧАННЯ СТУДЕНТІВ ІНТЕГРОВАНОЇ ТЕХНОЛОГІЇ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНИХ ПРОДУКТІВ

Анотація. У статті проведено всебічний огляд технік тест дизайну в контексті навчання студентів, які спеціалізуються на інтегрованих технологіях забезпечення якості програмних продуктів (Quality Assurance). Сучасний ринок IT-послуг постійно висуває нові вимоги до якості програмного забезпечення (ПЗ), включаючи підвищену надійність, безпеку, функціональність і зручність у користуванні. У зв'язку з цим, роль фахівців з QA значно зросла, а відтак і потреба у підготовці висококваліфікованих спеціалістів у цій галузі. На сьогоднішній день, багато навчальних програм у сфері інформаційних технологій зосереджуються на розробці програмного забезпечення, проте недостатньо уваги приділяють методам забезпечення його якості. Це призводить до того, що випускники не завжди володіють необхідними навичками для розробки ефективних тестових сценаріїв і використання сучасних технік тестування, що є критично важливим для виявлення та усунення дефектів на ранніх етапах життєвого циклу ПЗ. У статті детально розглянуто ключові техніки тест дизайну, серед яких: еквівалентний розподіл, аналіз граничних значень, таблиці прийняття рішень, парне тестування, тестування на основі зміни станів, а також техніки розробки сценаріїв для вичерпного та дослідницького тестування. Для кожної техніки наведено приклади її застосування у практичних завданнях, що дозволяє краще зрозуміти її ефективність та переваги. Особлива увага приділена питанням адаптації цих технік до навчання, спрямованого на підготовку студентів до реальних умов роботи в IT-індустрії. Це дозволить забезпечити студентів не лише теоретичними знаннями, але й практичними навичками, які є критично важливими для роботи у динамічному середовищі сучасних Agile та DevOps команд. У висновках підкреслюється, що впровадження технік тест дизайну в навчальні програми сприятиме формуванню у студентів системного мислення, аналітичних здібностей та вмінь працювати з сучасними інструментами забезпечення якості ПЗ. Отже, розширення освітніх програм шляхом інтеграції сучасних технік тест дизайну є необхідним кроком для підготовки конкурентоспроможних спеціалістів, здатних відповідати на виклики сучасного IT-ринку.

Ключові слова: програмні продукти, освітній процес, технології тест дизайну, тест-кейси, фреймворки, продуктивність.

Вступ. Постановка проблеми в загальному вигляді та обґрунтування її актуальності В сучасному інформаційному світі технології підкорюють все більше нових вершин, а тестування програмного забезпечення є неодмінною і необхідною складовою в процесі їх розробки. Велика кількість різних програм потребують ефективних підходів до тестування для забезпечення їхньої надійності та стабільності. В умовах стрімкого розвитку IT-індустрії, важливість підготовки висококваліфікованих спеціалістів у сфері забезпечення якості ПЗ є надзвичайно актуальною.

Одним із ключових аспектів у підготовці майбутніх спеціалістів є опанування технік тест дизайну. Тест дизайн є методологією створення тестових сценаріїв, що дозволяють ефективно оцінити працездатність програмного продукту, мінімізуючи ризики виявлення

критичних дефектів на пізніх етапах розроблення. Техніки тест дизайну допомагають забезпечити повноту та точність тестування, що є основою для підвищення якості ПЗ.

Проте, попри значну роль тест дизайну в процесі забезпечення якості, його інтеграція в навчальні програми закладів вищої освіти ще не є достатньо розробленою. Багато студентів, навіть після завершення курсів з програмування, не володіють достатніми навичками для розроблення тестових сценаріїв і використання ефективних технік тестування. Це створює розрив між вимогами ринку праці та реальними знаннями випускників.

Актуальність дослідження визначається необхідністю інтеграції технік тест дизайну у процес навчання студентів, що дозволить підвищити їх конкурентоспроможність на ринку праці та забезпечить високий рівень якості розроблення програмних продуктів. Ця проблема набуває особливої значущості в контексті навчання інтегрованих технологій забезпечення якості ПЗ, що охоплюють усі етапи життєвого циклу розроблення програмного забезпечення.

Аналіз останніх досліджень і публікацій. Проблеми підготовки студентів-програмістів до тестування програмного забезпечення в реальних виробничих умовах розглядали, такі науковці як, С.П. Альошин, О.Гура, А.М. Гафіяк, О.О. Бородіна.

І. І. Рудик та Б. В. Шевчук вважають, що забезпечення якості програмного продукту – одне із основних завдань розробників [4, с.188].

Закордонні науковці Маниш Кумар, Хазарибаг, Джаркханд, розглядають скорочення тестових випадків за допомогою систем таблиць рішень [13].

На думку О. Мелкозерової, В. Гайкової, С. Малахова універсальний підхід до тестування ускладнюється великими обсягами даних, наявністю різноманітних методик та інструментів [3, с. 38].

М. О. Пташинський, О. О. Фукс, У. Б. Марікуца описуючи та порівнюючи сучасні методи тестування продуктивності вебдодатків вказують на вирішальну роль сучасних методів тестування продуктивності вебдодатків у забезпеченні надійності та масштабованості вебдодатків у сучасному цифровому середовищі. Таким чином, проєктування тестів має бути свідомим заняттям, щоб намітити траєкторію тестування, адже розроблений дизайн буде виступати як керівництво для тестової команди, оскільки способи розробки тестових прикладів для системи, що тестується, нескінченні.

Мета написання статті: дослідити різні методології навчання технік тест дизайну, що дозволить майбутнім фахівцям не лише опанувати основні принципи тестування, але й ефективно застосовувати їх у практичних ситуаціях, що, своєю чергою, сприятиме підвищенню якості та надійності програмних продуктів. Для досягнення поставленої мети необхідним є виконання таких завдань:

1. Провести аналіз сучасних технік тест дизайну та визначити їх сильні і слабкі сторони.
2. Визначити вплив використання конкретних технік на якість програмного продукту.
3. Сформувати висновки та рекомендації щодо застосування технік тест дизайну в процесі навчання студентів інтегрованої технології забезпечення якості програмних продуктів.

Подання основного матеріалу дослідження. Яке б тестування не було детальним, але гарантувати відсутність всіх помилок неможливо, іншими словами тест будь-якої програми має бути неповним. Зрозумівши це, на перший план виходить важливий етап в тестуванні, такий як розроблення тест-кейсів, який покриває функції ПЗ наскільки це можливо [9, с. 43].

Тест дизайн (Test Design) – це процес розроблення вхідних даних, на основі яких створюються тестові кейси і в результаті чого відбувається ефективне тестування програмного забезпечення. Тест дизайн є одним із початкових етапів тестування програмного забезпечення, на якому плануються, проєктуються і створюються тестові випадки, відповідно до визначених раніше критеріїв якості і людських знань про тестування [15, с. 60-61].

Основними перевагами тест дизайну є:

- *ефективніше покриття тестами* – під час розробки тестових сценаріїв потрібно визначити конкретні сценарії та умови які потрібно протестувати, а це забезпечує високий рівень охоплення, що збільшує ймовірність виявлення дефектів;

- *раннє виявлення дефектів* – тестування за допомогою тест-дизайну допомагає виявити дефекти на ранній стадії розробки та дозволяє своєчасно вирішити проблему мінімізуючи вплив дефектів на пізніших етапах;
- *ефективне тестування* – тест дизайн спрямований на оптимізацію процесів тестування фокусуючись на основних аспектах уникаючи зайвих чи непотрібних кейсів, а це призводить до зменшення часу та коштів на процес розробки;
- *перевірка вимог* – порівняння тестових сценаріїв із системними вимогами допомагає переконатися, що система відповідає поставленим вимогам і працює так, як необхідно;
- *висока якість програмного забезпечення* – використання методів тестового дизайну дозволяє мінімізувати ризики, що своєю чергою призводить до задоволення клієнтів та зменшення витрат на обслуговування і підтримку [16].

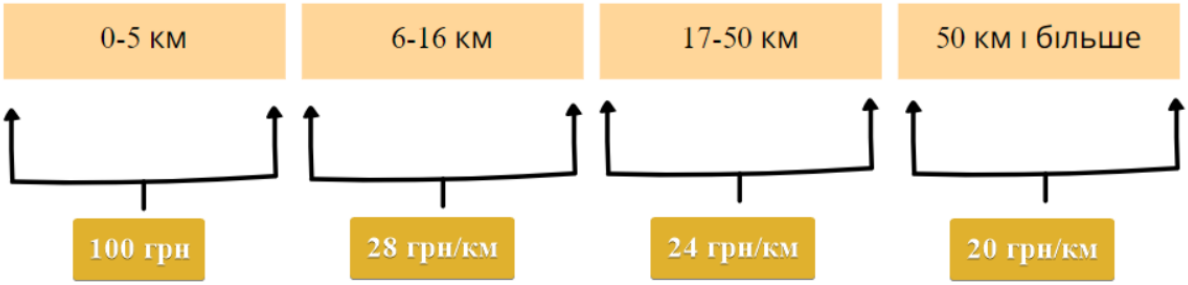
Equivalence Partitioning (Еквівалентний розподіл) – це метод створення тестових сценаріїв, який спрямований на ефективне та економічне тестування програми чи системи [10].

Група тестів із однаковим результатом утворюють клас еквівалентності із такими критеріями:

- вони перевіряють одне й те саме;
- якщо один тест виявляє помилку, інші тести з групи також будуть із цією помилкою і навпаки [8, с. 128].

Щоб легше зрозуміти, як ця техніка тест дизайну працює, розглянемо тарифікацію в таксі через призму еквівалентних класів. Уявимо, що тарифи в таксі встановлюються залежно від відстані подорожі (рис. 1):

- *0-5 км* – базовий тариф, 100 грн;
- *6-16 км* – 28 грн/км;
- *17-50 км* – 24 грн/км;
- *50 км і більше* – 20 грн/км.



Аналіз класів еквівалентності на прикладі тарифікації в таксі

Щоб не створювати тестові сценарій на кожен кілометр подорожі, потрібно вибрати одне або декілька значень з кожної групи і протестувати використовуючи дані значення. В нашому випадку:

- if (distance == 3) price = 100 грн;
- if (distance == 10) price = 28 грн/км;
- if (distance == 48) price = 24 грн/км;
- if (distance == 103) price = 20 грн/км.

Якщо тест виконаний успішно для одного з представників групи, то програма буде вірно працювати і для інших значень цієї групи. Іншими словами, тестування еквівалентного класу дозволяє визначити, на прикладі одного значення, чи програма правильно опрацьовує всі подібні дані з цього класу, а це своєю чергою спрощує тестування і робить його більш ефективним. [11, с. 40-54]

Boundary Value Analysis (Аналіз граничних значень) стосується визначення тестових сценаріїв, які перевіряють крайні точки кожного класу еквівалентності, тобто місце де відбувається перехід між класами і де, як правило, зосереджені помилки [12, с. 453-454].

Щоб використовувати тестування граничних значень, потрібно притримуватися таких кроків:

- визначити класи еквівалентності;
- уточнити межі для кожного еквівалентного класу;
- створити тест-кейси для кожного граничного значення, обираючи сценарій на самій межі (один трохи нижче і ще один вище межі).

Використання такого підходу дозволяє перевірити різні сценарії і допомагає виявити можливі дефекти на межі кожного еквівалентного класу [11, с. 29].

Вище проаналізовані класи еквівалентності на прикладі тарифікації в таксі, а для цього прикладу розглянемо граничні значення (рис. 2):

{-1, 0, 1}, {4, 5, 6}, {15, 16, 17}, {49, 50, 51}.

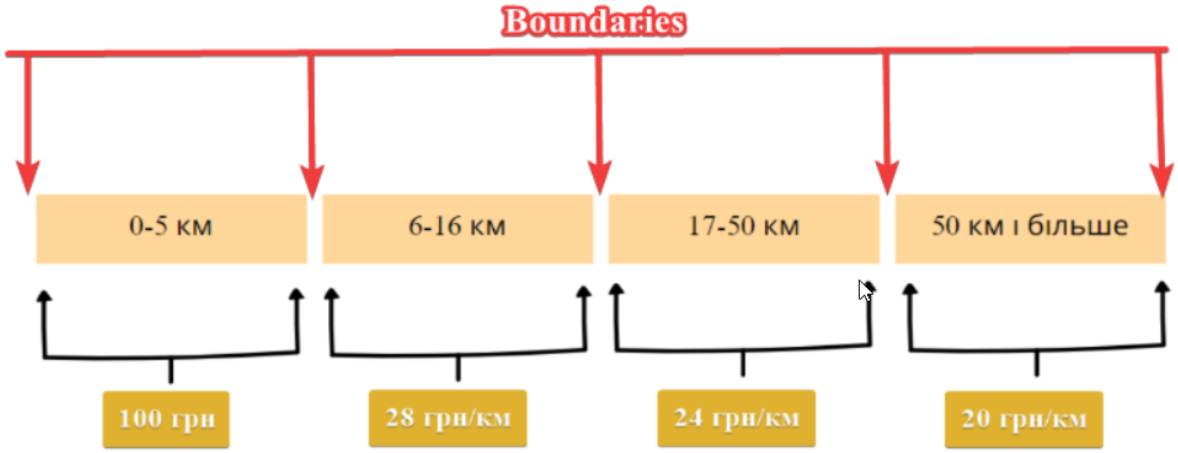


Рис. 2. Аналіз граничних значень на прикладі тарифікації в таксіРис. 1.

Під час тестування еквівалентних і граничних значень обов'язково потрібно пам'ятати про сценарії з негативним результатом, тобто ситуація коли введено недійсні дані і програма повинна їх опрацювати в залежності від поставлених умов.

Decision Table Testing (Таблиці прийняття рішень) – це техніка розробки тестових сценаріїв для перевірки різних комбінацій вхідних даних, які відображені у спеціальній таблиці рішень. За допомогою цієї техніки тест дизайну можна оцінити, як програма реагує на різні поєднання вхідних даних.

Для того щоб побудувати таблицю рішень, потрібно виконати такі кроки:

- визначити умови, які впливають на прийняття рішень (обирати потрібно ті умови, які можуть виникнути або не виникнути);
- визначити всі можливі дії, які відповідають різним умовам;
- розрахувати всі можливі комбінації;
- заповнити таблицю рішень, вказуючи дії, які слід вжити в кожній конкретній комбінації умов [13, с. 164-167].

Розглянемо конкретний приклад створення таблиці на прикладі знижок для клієнтів в службі таксі. У нас є такі умови:

1. Є два типи клієнтів «постійні» і «одноразові».
2. Кількість поїздок:

- менше 5;
- від 5 до 10;
- 10 і більше.

3. Спосіб оплати:

- готівка;
- безготівковий розрахунок.

Тепер розглянемо всі можливі дії з даними умовами (таблиця 1):

1. якщо клієнт з кількістю поїздок менше 5 і оплату проводить готівкою, то такий клієнт немає знижки;
2. якщо клієнт з кількістю поїздок від 5 до 10 і оплату проводить готівкою, то у такого клієнта знижка 10 %;
3. якщо клієнт з кількістю поїздок від 10 і більше а оплату проводить готівкою, то у такого клієнта знижка 10 %;
4. якщо клієнт проводить безготівковий розрахунок, то у клієнта додатково надається знижка 2% незалежно від кількості поїздок.

Таблиця 1

Таблиця рішень на прикладі знижок для клієнтів в службі таксі

Тип клієнта	Кількість поїздок	Спосіб оплати	Дія
разовий	менше 5	готівка	без знижки
потенційно постійний	від 5 до 10	готівка	знижка 10%
постійний	10 і більше	готівка	знижка 15%
разовий	менше 5;	безготівковий	знижка 2 % за безготівковий розрахунок
потенційно постійний	від 5 до 10	безготівковий	знижка 10% + 2 % за безготівковий розрахунок
постійний	10 і більше	безготівковий	знижка 15% + 2 % за безготівковий розрахунок

Розглянемо ще один приклад. Наприклад все таж служба таксі дає додаткову знижку студентам і пенсіонерам. За вимогами дана таблиця буде мати дві умови, кожна з яких має два значення або «Так» або «Ні» (табл. 2).

Таблиця 2

Таблиця рішень на прикладі додаткових знижок для клієнтів служби таксі

Умови	Правило 1	Правило 2	Правило 3	Правило 4
Студент?	Так	Так	Ні	Ні
Пенсіонер?	Так	Ні	Так	Ні
Дія				
Знижка	60 %	30 %	40%	0

У таблиці 1 та 2, відображаються всі можливі комбінації умов та визначають відповідні дії для кожної комбінації. Кожне правило своєю чергою є причиною запуску дії. Також потрібно зауважити, що в таблиці можна задати більше чим одну дію.

Таким чином, ця техніка допомагає під час розробки тестових сценаріїв, коли система повинна реалізувати складні бізнес-процеси та перевірки правильності виконання поставлених умов [11, с.67-76] .

PairWise Testing (Парне тестування) є економічною альтернативою тестування всіх можливих комбінацій набору змінних. Під час використання цієї техніки тест дизайну створюється набір тестових кейсів, який охоплює всі можливі варіанти пари значень обраних тестових даних. Процес тестування розпочинається з вибору значень для вхідних даних, а потім ці значення переставляються таким чином, щоб забезпечити покриття всіх можливих пар [6, с. 183].

Розглянемо приклад тестування з трьома факторами: A, B, C. Кожен фактор може мати три значення (відповідно: A1, A2, A3; B1, B2, B3; C1, C2, C3). Для того щоб зменшити кількість можливих комбінацій для тестування створюємо таблицю PairWise (табл. 3). Таким

чином покрилися всі можливі комбінації, але у такому випадку використовуємо лише 6 тестів.

Таблиця 3

Приклад попарного тестування з трьома факторами

Test	Фактор A	Фактор B	Фактор C
1	A1	B1	C1
2	A2	B2	C2
3	A3	B3	C3
4	A1	B2	C3
5	A2	B3	C1
6	A3	B1	C2

Розглянемо ще один приклад попарного тестування на прикладі чотирьох параметрів: *connection type* – тип підключення, *date transfer speed* – швидкість передачі даних, *protocol* – протокол і *encryption* – шифрування. Кожен з цих параметрів має по три можливих значень (табл. 4).

Таблиця 4

Приклад попарного тестування з чотирма параметрами

Connection type	Date transfer speed	Protocol	Encryption
Wi-Fi	Низька	TCP	Немає
Ethernet	Середня	UDP	WEP
Bluetooth	Висока	HTTP	WPA2

В цьому прикладі якщо перевіряти всі можливі комбінації чотирьох параметрів, то в нас вийде 3*3*3*3=81. Ця кількість перевірок займе чималий час, тому за допомогою парного тестування є можливість зменшити кількість тестів. Для цього потрібно виконати такі кроки:

1. вибрати перший параметр із списку можливих значень (у нашому випадку візьмемо *Connection type* з параметром Wi-Fi);
2. створювати пари, вибираючи кожен інший параметр з можливих значень (табл. 5);

Таблиця 5

Створені пари для першого параметру Wi-Fi

	Connection type	Date transfer speed	Protocol	Encryption
1	Wi-Fi	Низька	TCP	Немає
2	Wi-Fi	Середня	UDP	WEP
3	Wi-Fi	Висока	HTTP	WPA2

3. повторити цей процес для кожного значення. За допомогою цього процесу є можливість створити пари параметрів, які охоплять всі можливі комбінації вхідних параметрів. У цьому прикладі, так, як кожна з перших трьох комбінацій має по три варіанти інших параметрів, отримаємо 9 унікальних парних комбінацій (табл. 6).

Таблиця 6.

Створені пари для всіх параметрів

	Connection type	Date transfer speed	Protocol	Encryption
1	Wi-Fi	Низька	TCP	Немає
2	Wi-Fi	Середня	UDP	WEP
3	Wi-Fi	Висока	HTTP	WPA2
4	Ethernet	Середня	HTTP	Немає
5	Ethernet	Висока	TCP	WEP
6	Ethernet	Низька	UDP	WPA2
7	Bluetooth	Висока	UDP	Немає
8	Bluetooth	Низька	HTTP	WEP
9	Bluetooth	Середня	TCP	WPA2

Ці приклади показують, що можна скороти кількість тестів і в той же час якісно перевірити програмне забезпечення в різних умовах вхідних даних. За допомогою парного тестування є можливість виявити потенційні дефекти на етапі розробки.

State Transition Testing (Тестування зміни станів) – техніка тест дизайну призначена для покриття різних переходів між станами, включаючи допустимі і недопустимі переходи [16]. Для розуміння цієї техніки проаналізуємо всі кроки:

1. *Визначення станів.* Припустимо, що потрібно тестувати програму управління користувачами, в якій кожний користувач може мати три стани: *«Active»*, *«Deleted»*, *«Waiting»*:
   - *визначення подій* – їх потрібно визначити, так як вони впливають на стан системи. У цьому випадку дії можуть бути такі:
   - *«Реєстрація нового користувача»*;
   - *«активація облікового запису»*;
   - *«видалення користувача».*
2. *Створення діаграми переходів стану*, яка вказує на можливі переходи між станами в залежності від подій (рис. 3).

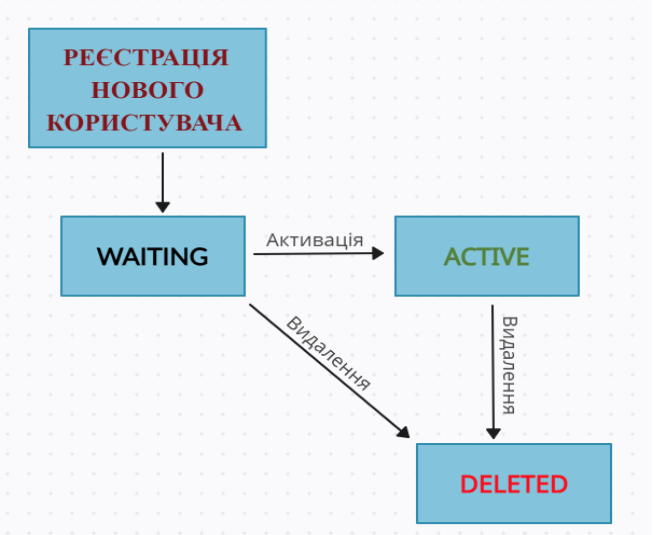


Рис. 3. Діаграма переходів стану

3. *Створення тестових кейсів* на основі діаграми:
   - *Test 1:* після реєстрації у користувача статус *«Waiting»*;
   - *Test 2:* статус *«Active»*, після дії – активація облікового запису;
   - *Test 3:* статус *«Deleted»*, після дії – видалення зі статусу *«Waiting»*;
   - *Test 4:* статус *«Deleted»*, після дії – видалення зі статусу *«Active»*.
4. *Проходження тестів.*

Техніка тестування зміни стану є ефективним методом для знаходження помилок, які система може зазнавати під час зміни стану. За допомогою цієї техніки можна забезпечити високий рівень покриття і детально оцінити реакцію на різні вхідні дані та умови використання.

Exhaustive Testing (Вичерпне тестування) – це техніка тест дизайну яка означає, що кожна можлива комбінація вхідних даних перевіряється для виявлення помилок. Однак цей підхід є не практичним через велику кількість можливих перевірок, які ускладнюють процес або навіть роблять його неможливим. Його використання може бути неефективним і вимагати великих ресурсів, в той же час не завжди гарантує впевненість в якості [3, с. 37]. Вичерпне тестування здебільшого використовується до програмного забезпечення, помилки в якій можуть викликати втрату грошей, критичних даних чи навіть життя і т.п. Тому до

такого програмного забезпечення використовується даний тип тестування, де необхідна максимальна впевненість у роботі системи.

Вичерпне тестування використовують у таких сферах:

1. *Авіація і космічні системи* – для тестування систем управляння польотами, систем контролю двигунами тощо.
2. *Медичне обладнання* – в сфері медицини активно використовують вичерпне тестування, так як від нього може залежати здоров'я і життя людей.
3. *Банківська сфера* – у цій сфері ряд критичних функцій, які можуть призвести до втрати коштів і краху всієї фінансової системи, тому надважливо перевірити всі можливі варіанти.
4. *Ядерна енергетика* – тут і без слів зрозуміло, що без вичерпного тестування не обійтися.

Exploratory Testing (Дослідницьке тестування) – це тестування методом вільного пошуку або іншими словами тестування без спроектованих раніше тестів, але керуючись певного плану, сценарію. Потрібно відмітити, що цей сценарій не є фіксованим але він може змінюватися та розширюватися в процесі розробки програмного забезпечення. Основною метою такого підходу є вивчення функцій та виявлення можливих проблем на ранній стадії.

Дослідницьке тестування необхідно використовувати у таких випадках:

- якщо не має вимог або вони часто змінюються;
- на ранніх етапах розробки, коли частина функцій вже працює, а інша ще розробляється;
- коли йде активне розроблення;
- коли всі підходи вже вичерпали себе;
- коли цілі наступної ітерації змінюються в залежності від результатів попередньої ітерації [10].

Error Guessing (Вгадування помилок) – це метод, під час якого процес перевірки відбувається на підставі досвіду і знань тестувальника про те, які недоліки часто виникають. Замість того, щоб випадково виявляти помилки, тестувальник використовує свій експертний досвід, щоб звернути увагу саме на проблемних місцях, що допомагає більш ефективно тестувати програмне забезпечення в процесі розробки.

Важко описати саму процедуру для техніки вгадування помилок, так як це здебільшого інтуїтивний процес, але основна ідея полягає в тому щоб скласти список можливих ситуацій, які можуть призвести до помилок, а потім на основі цього списку написати тест кейси. Оскільки описати таку процедуру важко, найкращою альтернативою буде навести приклад [14 с. 95-96]. В процесі тестування поля *email* під час реєстрації можна використати таку техніку для перевірки реакції системи на різні валідаційні помилки, наприклад:

- введення електронної пошти з неправильним форматом (*error@.com* чи *error@com*);
- введення спеціальних символів (*error$@example.com*);
- введення *email* без символу «@».

Звісно, це лише частина сценаріїв, використання яких дозволяє перевірити, як система веде себе в різних ситуаціях. Іншими словами використовуючи таку техніку потрібно перерахувати ті особливі випадки, які могли бути не враховані під час розроблення програми.

Ознайомившись із різними техніками тест дизайну виокремлюють такі основні задачі тест дизайну:

1. перед початком процесу тест дизайну важливим є конкретне визначення цілей тестування для того щоб орієнтуватися у виборі тестових сценаріїв та критеріїв покриття;
2. створення тестових сценаріїв та методів тестування в залежності від вимог;
3. розподіл функцій на класи еквівалентності та знаходження граничних значень;
4. аналіз того, наскільки тести покривають розроблені функції;

5. документування тестових сценаріїв;
6. аналіз результатів тестування.

Отже, тест дизайн є важливим етапом у процесі розроблення програмного забезпечення, на якому плануються, проектуються і створюються тестові сценарії у відповідності до вимог. Використання різноманітних технік тест дизайну допомагає виявити та вирішити потенційні помилки на ранніх стадіях розробки.

Для кращого засвоєння матеріалу та підготовки до роботи в умовах реальних проектів, студенти мають виконувати практичні завдання, які моделюють реальні сценарії тестування. Це можуть бути лабораторні роботи або курсові проекти, де студенти застосовують техніки тест дизайну для:

- Написання тестових сценаріїв та випадків (Test Cases) для заданих вимог.
- Виконання регресійного тестування для виявлення помилок після внесення змін у код.
- Аналізу результатів тестування та підготовки звітів про дефекти (Bug Reports).

Окрім теоретичних знань, студентам необхідно набути практичних навичок роботи з інструментами тестування, які широко використовуються в IT-індустрії:

- *Selenium, Cypress* – для автоматизації тестування вебдодатків.
- *JIRA, TestRail* – для управління тестами та відслідковування дефектів.
- *JMeter, Postman* – для тестування API та продуктивності.
- *SonarQube, Checkmarx* – для аналізу якості коду.

Навчання з використанням цих інструментів дозволяє студентам краще орієнтуватися у сучасних практиках тестування та бути готовими до роботи у проєктних командах. Використання кейс-стаді та моделювання реальних бізнес-ситуацій допомагає студентам зрозуміти важливість тестування в повному циклі розробки програмного забезпечення. Це може включати: аналіз вимог замовника та складання тестових планів, розробку сценаріїв тестування на основі користувацьких історій (User Stories), виконання тестування з урахуванням Agile та DevOps методологій. Важливо навчати студентів не лише виявляти помилки, але й оцінювати ефективність тестування. Це передбачає:

- Аналіз тестового покриття (Test Coverage) для визначення, наскільки повно перевіряється функціональність системи.
- Оцінку якості тестових сценаріїв, використовуючи метрики, такі як кількість знайдених дефектів та середній час на їх виправлення.
- Використання тестових звітів для обґрунтування необхідності додаткових ресурсів або змін у процесі розроблення.

Адаптація технік тест дизайну до навчальних програм є ключовим етапом підготовки студентів до реальних умов роботи в IT-індустрії. Впровадження цих технік у процес навчання сприятиме формуванню у студентів критичних навичок, необхідних для забезпечення якості ПЗ, що, у свою чергу, підвищить їхню конкурентоспроможність на ринку праці.

Висновки. Результати проведеного огляду технік тест дизайну свідчать про їх важливу роль у процесі забезпечення якості програмних продуктів. Опанування цих технік є необхідною складовою підготовки студентів, які навчаються за програмами, спрямованими на інтегровану технологію забезпечення якості ПЗ.

Інтеграція технік тест дизайну у навчальні програми дозволить не лише підвищити рівень професійної підготовки майбутніх фахівців, але й сприятиме формуванню у них критичного мислення та навичок системного підходу до розв’язування задач, пов’язаних із забезпеченням якості програмного забезпечення. Це, у свою чергу, сприятиме їхній конкурентоспроможності на сучасному ринку праці, де попит на таких спеціалістів продовжує зростати.

Таким чином, для досягнення високої ефективності навчального процесу, важливо розробляти та впроваджувати нові методики викладання, що включають використання технік тест дизайну у практичних завданнях. Це забезпечить не лише глибше розуміння теоретичних аспектів тестування, але й надасть студентам можливість здобути практичні

навички, необхідні для успішної професійної діяльності у сфері забезпечення якості програмних продуктів.

Подальші дослідження у сфері технік тест дизайну в контексті навчання інтегрованої технології забезпечення якості програмних продуктів плануються у вивченні можливостей інтеграції технік тест дизайну у контексті DevOps та Agile методологій. Це дозволить студентам краще розуміти сучасні підходи до розробки ПЗ, де забезпечення якості є невіддільною частиною всього життєвого циклу проєкту.

Список використаних джерел:

[1] Єрмоленко М., Шевчук Б. Значення автоматизації тестування у сфері програмування. Матеріали конференцій МНЛ. м. Біла Церква, 2022. С. 179-180.

[2] Закусило М., Шевчук Б. Онлайн інструменти та системи тестувальника. УДУ імені Михайла Драгоманова, 2024. С.95-98.

[3] Мелкозьорова Ольга, Валерія Гайкова, Сергій Малахов. Опис схеми API тестування програмного забезпечення. *Комп'ютерні науки та кібербезпека* 4. 2019. С. 38-45.

[4] Рудик І. І., Шевчук Б. В.: Потреба тестування програмного забезпечення. Матеріали конференції МНЛ, с.186-188. Біла Церква (2022).

[5] Шевчук Б., Закусило М. Порівняльний аналіз технології забезпечення якості програмних продуктів за допомогою верифікації та валідації. International scientific-practical conference "Science, education and technology: trends, challenges, prospects" 2024. С. 47-48.

[6] Bach James, and Patrick J. Schroeder. "Pairwise testing: A best practice that isn't." Proceedings of 22nd Pacific Northwest Software Quality Conference. 2004. P. 180-196.

[7] Bach James. "Exploratory testing explained." URL: http://www. satisfice. com/articles/et-article. Pdf (2003): 1-10.

[8] Cem Kaner, Jack Falk Hung, Quoc Nguyen. Testing Computer Software. Bibliyografya ve İndeks Wiley computer publishing. 1999. 496 p.

[9] Glenford J. Myers, Corey Sandler, Tom Badgett, Todd M. Thomas. The Art of Software Testing Second Edition. PublisherWileyEdition 2., Уайли. 2004.

[10] Halyna Zakharova Техніки тест-дизайну для «чайників». 2023. URL: https://dou.ua/forums/topic/44882/

[11] Lee Copeland. A Practitioner's Guide to Software Test Design Artech House Computing Library.2003. 300 p.

[12] Lewis, William E. Software testing and continuous quality improvement. Gunasekaran. Veerapillai, technical contributor.2004. 560 p.

[13] Manish Kumar, Hazaribag, Jharkhand Dr. From Knowledge Discovery to Implementation: Reducing Test Cases using Decision Table Systems. International Journal of Innovations & Advancement in Computer Science IJIACS ISSN 2347 – 8616 Volume 7, Issue 1 January 2018.

[14] Mayer J, Guderlei R. On random testing of image processing applications. In: 2006 Sixth International Conference on Quality Software (QSIC'06). IEEE; 2006:85-92.

[15] Paul Ammann George Mason Introduction to software testing. Cambridge University Press., USA. 2008. 273 p.

[16] Test Design Techniques: BVA, State Transition, and more Artem Golubev. TestRigor. 2023. URL: https://testrigor.com/blog/test-design-techniques-bva-state-transition-and-more/

OVERVIEW OF TEST DESIGN TECHNIQUES IN THE CONTEXT OF TEACHING STUDENTS IN THE INTEGRATED TECHNOLOGY OF SOFTWARE QUALITY ASSURANCE

Mykola Zakusylo, Borys Shevchuk

Abstract. The article provides a comprehensive review of test design techniques in the context of training students who specialize in integrated technologies for quality assurance of software products (Quality Assurance). The modern IT services market constantly puts forward new requirements for software quality, including increased reliability, security, functionality, and ease of use. In this regard, the role of QA specialists has increased significantly, hence the need to train highly qualified specialists. Today, many educational programs in the field of information technology focus on software development, but not enough attention is paid to methods of ensuring its quality. As a result, graduates do not always have the necessary skills to develop effective test scenarios and use modern testing techniques, which are critical for detecting and eliminating defects early in the software life cycle. The article details key test design techniques, including equivalent distribution, boundary value analysis, decision tables, paired testing, state transition testing, and scenario development techniques for exhaustive and exploratory testing. For each method, examples of its application in practical tasks are given, allowing you to better understand its effectiveness and advantages. Special attention is paid to the issue of adapting these techniques to training aimed at preparing students for actual working conditions in the IT industry. This will provide students with theoretical knowledge and practical skills critical for working in the dynamic environment of modern Agile and DevOps teams. The conclusions emphasize that introducing test design techniques into educational

programs will contribute to the formation of students' system thinking, analytical abilities, and the ability to work with modern software quality assurance tools. Therefore, expanding educational programs through integrating modern test design techniques is necessary for training competitive specialists capable of responding to the challenges of the modern IT market.

Keywords: software products, educational process, test design technologies, test cases, frameworks, productivity.

References (translated and transliterated)

[1] Iermolenko M., Shevchuk B. (2022) Znachennia avtomatyzatsii testuvannia u sferi prohramuvannia [The value of test automation in the field of programming]. Materialy konferentsii MNL. m. Bila Tserkva. P. 179-180. (In Ukraine)
[2] Zakusylo, Mykola, Shevchuk Borys (2024) Onlain instrumenty ta systemy testuvalnyka. [Online tester tools and systems] UDU imeni Mykhaila Dragomanova. P. 95-98. (In Ukraine)
[3] Melkozorova, Olha, Valeriia Haikova, Serhii Malakhov. (2019) Opys skhemy API testuvannia prohramnoho zabezpechennia. [Description of the software testing API scheme] Kompiuterni nauky ta kiberbezpeka 4. P. 38-45. (In Ukraine)
[4] Rudyk I. I., Shevchuk B. V. (2022) Potreba testuvannia prohramnoho zabezpechennia. [The need for software testing] Materialy konferentsii MNL, p.186-188. Bila Tserkva. (In Ukraine)
[5] Shevchuk B., Zakusylo M. (2024) Porivnialnyi analiz tekhnolohii zabezpechennia yakosti prohramnykh produktiv za dopomohoiu veryfikatsii ta validatsii.[Comparative analysis of software product quality assurance technology using verification and validation] International scientific-practical conference “Science, education and technology: trends, challenges, prospects”. P. 47-48. (In Ukraine)
[6] Bach, James, and Patrick J. Schroeder. (2004) "Pairwise testing: A best practice that isn’t." Proceedings of 22nd Pacific Northwest Software Quality Conference. P. 180-196. (In English)
[7] Bach, James. "Exploratory testing explained." [Online]. Available: http://www. satisfice. com/articles/et-article.pdf (2003): 1-10. (In English)
[8] Cem Kaner, Jack Falk Hung, Quoc Nguyen. Testing Computer Software. Bibliyografya ve İndeks Wiley computer publishing. 1999. 496 p. (In English)
[9] Glenford J. Myers, Corey Sandler, Tom Badgett, Todd M. Thomas. The Art of Software Testing Second Edition. PublisherWileyEdition 2., Уайли. 2004. (In English)
[10] Halyna Zakharova. Test design techniques for "teapots". 2023. [Online]. Available: https://dou.ua/forums/topic/44882/ (In Ukraine)
[11] Lee Copeland. A Practitioner's Guide to Software Test Design Artech House Computing Library. 2003. 300 p. (In English)
[12] Lewis, William E. (2004) Software testing and continuous quality improvement. Gunasekaran. Veerapillai, technical contributor. 560 p. (In English)
[13] Manish Kumar, Hazaribag, Jharkhand Dr. From Knowledge Discovery to Implementation: Reducing Test Cases using Decision Table Systems. International Journal of Innovations & Advancement in Computer Science IJIACS ISSN 2347 – 8616 Volume 7, Issue 1 January 2018. (In English)
[14] Mayer J, Guderlei R. On random testing of image processing applications. In: 2006 Sixth International Conference on Quality Software (QSIC'06). IEEE; 2006:85-92. (In English)
[15] Paul Ammann George Mason Introduction to software testing. Cambridge University Press., USA. 2008. 273 p. (In English)
[16] Test Design Techniques: BVA, State Transition, and more Artem Golubev. TestRigor. 2023. [Online]. Available: https://testrigor.com/blog/test-design-techniques-bva-state-transition-and-more/ (In English)