Науковий вісник НЛТУ України
Scientific Bulletin of UNFU
https://doi.org/10.36930/40330214
Article received 07.04.2023
Article accepted 25.04.2023
UDC 004.[01+02+05]:004.4
Correspondence author: Yu. I. Hrytsiuk yurii.i.hrytsiuk@lpnu.ua
ISSN 1994-7836 (print)
ISSN 2519-2477 (online)

П. Ю. Грицюк, А. В. Іванишин, Ю. І. Грицюк
Національний університет "Львівська політехніка", м. Львів, Україна

ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ПРОДУКТУ ЗА СТАНДАРТОМ IEEE 730-2014 В МЕЖАХ ЖИТТЄВОГО ЦИКЛУ РЕАЛІЗАЦІЇ ПРОЕКТУ

Проаналізовано наявні підходи до вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту, розглянуто та проаналізовано ефективність застосування відповідних стандартів для тестування реального веб-додатку, де звернуто увагу на етапи розроблення відповідних тестів, процеси забезпечення його якості та особливості розгортання. Проаналізовано настанови стандарту IEEE 730-2014, які дають можливість гарантувати якість програмного продукту, а також стандарт ISO/IEC 12207:2018, який забезпечує якість ПЗ протягом його життєвого циклу, а також особливості їх практичного застосування. Встановлено, що гарантія якості програмного продукту є обов'язковою для будь-якого ІТ-бізнесу, незалежно від того, чи це продукт для виконання повсякденних операцій, чи для споживачів критичної інфраструктури. Завдання перевірки якості, викладені в стандарті IEEE 12207:2018 для процесів життєвого циклу ПЗ, гарантують якість його розроблення та ефективність використання користувачами. З'ясовано, що стандарт IEEE 730-2014 описує SQA-процеси, які потрібно використовувати протягом усього життєвого циклу ПЗ. Наявність різних видів SQA-діяльності, закладених у відповідних стандартах, гарантують ПЗ належну його якість, що відповідатиме потребам замовника. Визначено ефективні SQA-процеси забезпечення якості продукту проекту, які потрібно не тільки виконувати, але й підтверджувати їх виконання, особливості вимірювання та відстежування відповідних процесів, правила розроблення заходів для управління ними та їх вдосконаленням, а також підходи до заохочення команди проекту використовувати SQA-процеси під час реалізації програмних проектів. Встановлено, що для досягнення цілей реалізації проекту потрібно запроваджувати координацію та верифікацію продукту проекту, його підтвердження та перегляд, аудит та інші вказівки стандарту ISO/IEC 12207:2018. Виходячи з цих вказівок, було сформовано структуру SQA-процесів для веб-додатку "Інтернет-Банкінг", який було протестовано в ході виконання дослідження. З'ясовано, що кожен винятковий програмний проект має свої особливості запровадження процесів забезпечення якості продукту проекту. В банківському додатку надзвичайно важливо проводити тестування ПЗ на кібер-безпеку, цілісність клієнтських даних, спостережність транзакцій, що і було описано в даній роботі. Після виконання всіх SQA-процесів продукт проекту "Інтернет-банкінг" було представлено безпосередньому користувачу. Розгортання ПЗ містило дії, необхідні для того, щоб зробити його доступним для виявлення дефектів під час реального його використання у виробничих умовах. Виявлено, що ефективна система контролю якості ПЗ забезпечує кращий досвід роботи з безпосередніми користувачами і збільшує тривалість його безвідмовної роботи. Водночас, навіть поодинокі дефекти, виявлені в роботі ПЗ, можуть вплинути на ключові бізнес-процеси замовника, що призведе до значних втрат прибутку. Запобігання цьому – шанс створити імідж бренду компанії-розробнику ПЗ, якій довіряють.

Ключові слова: стандарти якості програмних продуктів; гарантія якості програмного забезпечення; тестування; SQA-діяльність; специфікація вимог; вимоги замовника; контроль якості.

Вступ / Introduction

Сучасний рівень розвитку комп'ютерних технологій, проектування архітектури програмного забезпечення (ПЗ) та подальше його розроблення характеризується постійним зростанням складності не тільки його програмних компонент, але й концепцій та ідей його функціонування, що знаходяться в основі цих технологій. У цих складних умовах особливо важливим є об'єктивне оцінювання якості як остаточного програмного продукту, так і належного його стану на кожному етапі життєвого циклу. Для забезпечення якості ПЗ та подальшого його обслуговування використовують відповідні стандарти.

Стандарт IEEE Std 730 [23] був вказівником для фахівців із гарантії якості ПЗ (англ. *Software Quality Assurance*, SQA) з моменту його публікації у 1979 році. Його метою було забезпечення єдиних мінімально прийнятних вимог до SQA-процесів для підтримки проекту розроблення ПЗ. Хоча попередні його версії надавали

Інформація про авторів:
Грицюк Павло Юрійович, магістр, магістрант, кафедра інформаційно-вимірювальних технологій.
Email: pavlo.hrytsiuk.mmtias.2022@lpnu.ua
Іванишин Алла Василівна, канд. техн. наук, доцент, кафедра інформаційно-вимірювальних технологій.
Email: alla.v.hunkalo@lpnu.ua; https://orcid.org/0000-0002-3302-7889
Грицюк Юрій Іванович, д-р техн. наук, професор, кафедра програмного забезпечення.
Email: yurii.i.hrytsiuk@lpnu.ua; https://orcid.org/0000-0001-8183-3466

Цитування за ДСТУ: Грицюк П. Ю., Іванишин А. В., Грицюк Ю. І. Забезпечення якості програмного продукту за стандартом IEEE 730-2014 в межах життєвого циклу реалізації проекту. Науковий вісник НЛТУ України. 2023, т. 33, № 2. С. 101–117.
Citation APA: Grytsiuk, P. Yu., Ivanyshyn, A. V., & Hrytsiuk, Yu. I. (2023). Quality assurance of software products in accordance with IEEE 730-2014 standard within the project implementation lifecycle. *Scientific Bulletin of UNFU*, 33(2), 101–117. https://doi.org/10.36930/40330214

команді проекту план їхньої SQA-діяльності, однак подальший перегляд розширював сферу її застосування стандартом ISO/IEC 12207:2018 [50] для процесів, визначених у структурі життєвого циклу проекту. Така зміна акцентів була узгоджена з процесом розроблення ПЗ і вдосконаленням вимог до нього, насамперед вимог до його функціоналу, планування архітектури та графіку виконання робіт, контролю за діями щодо забезпечення якості ПЗ та його обслуговування.

Розглядаючи потребу прийняття цього стандарту, регуляторні органи мають знати, що конкретне його застосування може бути охоплено одним або кількома стандартами IEEE (англ. *Institute of Electrical and Electronics Engineers* – Інститут інженерів з електротехніки та електроніки) або ANSI (англ. *American National Standards Institute* – Американський інститут національних стандартів). Тому гарантія якості ПЗ (SQA) охоплює ввесь життєвий цикл проекту [17, 21, 29, 41], в т.ч. й такі процедури як визначення вимог, проектування архітектури, написання програмного коду, контроль вихідного коду та його аналіз, конфігураційне управління, тестування, управління релізами та інтеграція продуктів. Також такий процес містить цілі, можливості, процедури, вимірювання та перевірки. Водночас, дотримання вимог зазначених стандартів дає змогу програмним продуктам повною мірою задовольнити вишукані потреби безпосереднього користувача та різні обмеження регуляторних органів. Так звані гнучкі методи розроблення ПЗ, які не так давно набули широкого застосування, постійно розвиваються. Однак, провідні ІТ-компанії визнають, що все ще існують невирішені проблеми, а також резерви вдосконалення процесу розроблення ПЗ в гнучкому середовищі [37, 67].

Тому розроблення наукового підходу до вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту, пошук резервів ефективного їх застосування дадуть змогу протестувати реальний веб-додаток стосовно відповідних тестів, особливостей його тестування та розгортання є актуальною проблемою сьогодення. Робота спрямована на удосконалення уже наявних підходів до вирішення даної проблеми шляхом застосування настанов стандарту IEEE 730-2014, які дають можливість гарантувати якість програмного продукту, а також стандарт ISO/IEC 12207:2018 [50], який забезпечує якість ПЗ протягом його життєвого циклу, а також особливості їх практичного застосування.

Об'єкт дослідження – вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту.

Предмет дослідження – методи і засоби вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту, резерви ефективного застосування яких дадуть змогу протестувати реальний веб-додаток стосовно розроблення відповідних тестів, особливостей його тестування та розгортання.

Мета роботи – проаналізувати наявні підходи до вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту, з'ясувати резерви ефективного застосування цих підходів під час тестування реального веб-додатку насамперед щодо розроблення відповідних тестів, особливостей його тестування та розгортання в середовищі безпосереднього користувача.

Для досягнення зазначеної мети необхідно вирішити такі *основні завдання дослідження*:

- проаналізувати стан дослідженості проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту;
- проаналізувати настанови стандарту IEEE 730-2014, які гарантують якість програмного продукту, а також стандарт ISO/IEC 12207:2018, який забезпечує якість ПЗ протягом його життєвого циклу;
- проаналізувати стандарт IEEE 730-2014, настанови якого описують SQA-процеси, тобто комплексну діяльність, яку використовують протягом усього життєвого циклу ПЗ;
- проаналізувати стандарт ISO/IEC 12207:2018, настанови якого дають можливість досягнути цілі реалізації проекту шляхом запровадження відповідних заходів щодо координації та верифікації продукту проекту;
- сформувати структуру SQA-процесів для реального веб-додатку, який потрібно протестувати в ході виконання дослідження;
- розгорнути продукт проекту "Інтернет-банкінг" в середовищі безпосереднього користувача для виявлення реальних дефектів;
- зробити висновки за результатами виконаного дослідження та надати рекомендації щодо їх практичного використання.

Аналіз останніх досліджень та публікацій. У роботі [65] автор розглядає наявні як на той час стандарти програмної інженерії, які стосуються процесу розроблення ПЗ. Він вважає, що цей процес можна вдосконалити в напрямі професійного його визнання, хоча професію інженера-програміста як такою не визнають [12, 62, 65]. Одним із елементів такого вдосконалення є створення сукупності стандартів для відповідальних практиків [66]. У дослідженні [65] автор описує стандарти, які стосуються організацій з розроблення стандартів (англ. *Standards Developing Organizations*, SDO), що працюють на принципах консенсусного розроблення [71]. Однак, співтовариства консенсусу бувають різними. Наприклад, Інститут інженерів з електротехніки та електроніки (англ. *Institute of Electrical and Electronic Engineers*, IEEE) зазвичай розробляє свої стандарти шляхом досягнення консенсусу технічних фахівців, яких самостійно обирають з членів організації [22, 23]. З іншого боку, міжнародні стандарти формують консенсусом розробок національних органів, тобто представників націй, які голосують, ймовірно, представляючи свої національні інтереси. Тому майже всі стандарти, які вони розробляють, призначені для добровільного прийняття; тобто ІТ-компанія без примусу ухвалює власне рішення – прийняти стандарт чи його ігнорувати. Це контрастує з нормативними стандартами, нав'язані процесами, подібними до закону, і обов'язковими стандартами, такими як військові стандарти, дотримання яких необхідно як передумову ведення бізнесу з домінуючим клієнтом. Автор роботи [65] вважає, що ІТ-компанії мають добровільно приймати міжнародні стандарти [30, 44], позаяк вони покращують як свою продукцію, так і її сприйняття на конкурентному ринку. Окрім цього, такі стандарти можуть значно покращити бізнес-процеси ІТ-компанії, даючи їй можливість робити свою продукцію економічно ефективніше.

У роботі [8] автор розглядає наявні стандарти процесу розроблення ПЗ, які забезпечують його якісні показники. Насамперед було проаналізовано: стандарт IEEE 730-2014 для забезпечення якості ПЗ [23]; стандарт ISO/IEC 12207 [32, 50, 57, 75] для встановлення спільної основи процесів розроблення ПЗ; стандарт IEEE 1012 [22] для його перевірки та підтвердження

(англ. *Verification and Validation*, V&V). Автор роботи [8] стверджує, що стандарт IEEE 730-2014-2014 [23] представляє вимоги, які охоплюють особливості ініціалізації, планування, контролю та виконання ПЗ і гарантія його якості (англ. *Software Quality Assurance*, SQA) для повного життєвого циклу реалізації програмного [3, 54]. Стандарт ISO/IEC 12207:2008 [32] забезпечує структуру, яка містить ввесь перелік процесів життєвого циклу ПЗ [30, 40, 43, 61]. Основна частина стандарту ISO/IEC 12207:2008 [32] і IEEE 1012-2012 [22] описують процеси, дії та завдання життєвого циклу ПЗ [4, 57, 75]. Стандарт IEEE 1012-2012 – це настанови, які визначають всі SQA-процеси протягом життєвого циклу ПЗ як дії V&V [22]. Стандарт вимагає застосування діяльності V&V для системних, програмних, апаратних, управлінських і адміністративних процесів [23, 61, 62].

У роботі [11] автори розглядають особливості гнучкого розроблення ПЗ. Наведено різні документи, пов'язані з реалізацією цього процесу, які стосуються низки дослідницьких напрямів, у т.ч. й застосування гнучких методів для розроблення критично важливого ПЗ для його безпеки, взаємозв'язок гнучкого розроблення з дизайном користувача та способи вимірювання потоку виконання робіт для економного розроблення ПЗ. Автори стверджують [11], що термін "гнучкі методи" існує вже з 2000-го року, тоді як основні концепції та більшість практик [66], пов'язаних із гнучким розробленням ПЗ, існують набагато довше. Фактично серед багатьох дослідників ще й досі немає повної згоди щодо визначення гнучкого розроблення ПЗ [45], але, безумовно, гнучкі методи спрямовані на те, щоб задовольнити потребу пришвидшення цього процесу в середовищі мінливих вимог [31, 46, 47, 48]. Використання ітераційного процесу розроблення ПЗ є загальним для всіх гнучких методів, і, як правило, різні версії відповідного інструментарію [52] часто випускають для потенційних клієнтів. При цьому заохочують тісну (бажано на місці) співпрацю команди розробників з замовниками, а зміна вимог є прийнятною, навіть бажаною частиною процесу реалізації програмного проекту. Іншими типовими відмінностями від негнучких підходів є зосередження уваги на невеликих самоорганізованих командах, які здатні реалізувати ідею постійного вдосконалення архітектури ПЗ, його тестування та впровадження, а також постійна інтеграція членів команди у гнучкий виробничий процес. Окрім цього, управління знаннями при гнучкому розробленні ПЗ має тенденцію бути негласним, а не базуватися на вичерпній документації, а особисте спілкування з клієнтом вважається кращим засобом встановлення тісних взаємовідносин.

У роботі [64] автори розглядають особливості процесу реалізації комплексного програмного проекту, в якому відбувається адаптація гнучких методів розроблення ПЗ. Вони вважають, що парадигма гнучкого розроблення ПЗ стає все більш популярною в останнє десятиліття, оскільки вона вимагає менших витрат, кращої продуктивності членів команди та якості самого ПЗ, а також кращого задоволення команди розробників від виробничої діяльності. Водночас, управління ланцюгом поставок (англ. *Supply Chain Management*, SCM) – це складний виробничий процесу реалізації програмного проекту. Через обсяг виконуваних робіт і певні невизначеності, складні та нестабільні вимоги до продукту проекту процес розроблення ПЗ неможливо реалізувати тільки за допомогою передбачуваних моделей. Гнучкі методології націлені насамперед на вирішення таких проблем, які містять зміни та невизначеності, і є адаптивними, а не прогнозними. Спосіб впровадження гнучкого розроблення ПЗ істотно впливає на успіх запровадження змін у команді його розробників. У своєму дослідженні автори [64] аналізують гнучкої методології розроблення ПЗ та підходи до управління його якістю, які успішно використовують при реалізації навіть складних програмних проектів [1, 53]. Це додатково демонструє, як подолати ризики та бар'єри [73, 2] на кожній стадії розроблення ПЗ під час реалізації таких складних інноваційних проектів. Дослідження також містить набір вказівок щодо прийняття командою розробників гнучкої методології, їхньої комбінації з іншими технологіями під час складних проектів розроблення ПЗ [64, 68]. Ці висновки мають наслідки для інженерів з якості ПЗ та менеджерів, які його контролюють гнучкими методами.

У роботі [3] автори розглядають проблему пошуку гнучких методологій розроблення ПЗ для забезпечення його якості. Вони вважають, що використання гнучких методологій значно покращує якість ПЗ, не розглядаючи його якість як задоволення потреб клієнтів, а також його перевірку та підтвердження (V&V). Гнучке розроблення ПЗ інноваційно відкриває нові горизонти в забезпеченні його якості, це крок попереду традиційних методів розроблення, спрямоване на задоволення мінливих вимог клієнтів до моменту остаточного впровадження продукту проекту. У своєму дослідженні [3] автори системно аналізують гнучкі методології розроблення ПЗ з точки зору забезпечення його якості та наводять методику розуміння подібності в різноманітних гнучких процесах. Розуміння гнучкої методології розроблення ПЗ подано з різних точок зору: теоретичної, наукової та контекстної [1, 53, 60]. Незважаючи на те, що процес гнучкого розроблення ПЗ значно розширив межі реалізації програмного проекту, які можна помітити в контексті актуальності продукту проекту або своєчасної його доставки замовнику. Однак, мало яка команда розробників ПЗ дотримується цієї інноваційної методології через відсутність чіткого розуміння основних її особливостей.

У своєму дослідженні [3] автори детально визначили особливості гнучкої методології розроблення ПЗ та розкрили її значення порівняно з традиційними підходами з можливістю забезпечення якості ПЗ. Вони навели техніку, яка розкриває гнучкі методи, а саме, Scrum-методи [6, 2], кристалічну методологію (англ. *Crystal Methodology*), бережливе розроблення (англ. *Lean Development*), Канбан (англ. *Kanban*), екстремальне програмування (англ. *Extreme Programming*) тощо з використанням нового підходу, а також результати обговорення побоювань щодо впровадження гнучких методологій у виробничий процес. Ідея запропонованої технології полягає в тому, щоб вивчити кожну сучасну гнучку методологію, зосередившись на основних її цінностях і практиках [66], і представити на широкий загал ІТ-спільноти для внесення додаткових змін у гнучкі програмні проекти з метою забезпечення якості ПЗ. Ідея полягає в тому, щоб використати знання експертів-практиків для вдосконалення гнучкого розроблення ПЗ, що призведе до кращої його якості. Також в роботі [3] проаналізовано проблеми та суперечки навіть прихильни-

ків гнучких методологій, які є сірими зонами такого підходу, пов'язаного з інноваційним мисленням і вартістю реалізації таких проектів тощо. Також робота завершується обговоренням викликів, що очікують гнучку методологію розроблення ПЗ та рекомендацій щодо їх вирішення.

Хоча гнучкі методи розроблення ПЗ постійно розвиваються, проте, провідні ІТ-компанії визнають, що все ще існують невирішені проблеми, а також резерви вдосконалення процесу розроблення ПЗ в гнучкому середовищі [37, 67]. Наприклад, критичне для безпеки ПЗ все ще потребує строгого підходу, незалежно від того, чи використовують для цього традиційні та гнучкі методи. В роботі [11] описано досвід застосування гнучких методів розроблення ПЗ з відкритим вихідним кодом, надаючи користувачу критично важливий для безпеки програмний продукт. Автори визнають, що гнучкі методи, як правило, вважаються непридатними для ПЗ, критичного для безпеки, але, використовуючи свій досвід розроблення набору хірургічних інструментів із зображенням (англ. *Image Guided Surgical Toolkit*, IGSTK) [10, 14, 26], продовжують надавати докази, які спростовують це припущення. Спершу отримавши набір найкращих практик [66], багато з яких конкретно пов'язані з критично важливими потребами безпеки, обговорювана команда розробників змогла використати та розширити гнучкі методи для критичних потреб безпеки IGSTK [13, 14, 25]. Водночас, у роботі [8] і стандартах [32, 51, 28] описано додаткові практики, використані для управління вимогами та відстеження стану їх виконання [24, 25, 26], практика "безпеки за проектом", використання безперервної інтеграції та тестування [32, 45, 39], що сприяє підтримці інструментів, а також використання зворотного проектування моделей і подальшої їх перевірки для розроблення архітектури ПЗ. Загалом, на підставі свого досвіду автори [11] стверджують, що гнучкі методи, доповнені "потрібною часткою церемонії", справді можна використати для розроблення ПЗ, критичного для безпеки.

Гнучкі методи підкреслюють сучасні можливості розроблення та використання ПЗ. Незважаючи на це, користувачі все ще очікують високого рівня зручності використання ПЗ. Також часто буває так, що проектування користувальницького інтерфейсу є окремою діяльністю від розроблення функціональних можливостей ПЗ [76], і його часто виконують окремою особою або командою. Наприклад, в роботі [11] автори розглядають цю ситуацію, використовуючи спостереження за зрілою командою Scrum у великій ІТ-компанії [6, 2]. У статті описано спостереження за роботою двох окремих команд, одна з яких займається розробленням ПЗ, а інша – можливостями користувацького досвіду (англ. *User Experience*, UX). Дослідники використали свої результати, щоб скласти картину взаємодії між двома групами. Практики їхньої взаємодії [66], використані для інтеграції роботи двох команд, неможливо пояснити розглядом процесів Scrum [64] або можливостями UX, тому автори [11] приходять до висновку, що вони пов'язані з "розташованим характером" двох команд. Під цим вони мають на увазі, що те, що впливає на взаємодію та формує її, значною мірою обумовлене їхніми окремими зобов'язаннями на рівні підрозділів. У роботі [11] також описано, що дві команди окремо працюють швидше, ніж спільно, і що ця співпраця в основному реалізується через артикуляційну роботу. Це означає, що розробники мали додаткові завдання, щоб переконатися в тому, що можливості UX було завершено.

У роботі [59] автори розглядають проблеми контекстуалізації гнучкого розроблення ПЗ. Вони навели контекстну модель розроблення програмно-інтенсивних систем, щоб управляти процесом впровадження і адаптацією сучасних практик гнучкого розроблення ПЗ [66]. Ця модель була визнана особливо корисною, коли контекст реалізації програмного проекту істотно відхиляється від "гнучкої очікуваної точки", тобто ідеальних умов, у яких виникла проблема гнучкого розроблення ПЗ, і де ця проблема швидше за все досягне критичного значення. Автори [59] вважають, що це стосується великих програмних систем, розподілених середовищ розроблення [67], критично важливих для безпеки таких систем, систем, що потребують нової архітектури, або систем із неортодоксальною бізнес-моделлю чи моделлю системи управління [1, 53, 60, 68].

У роботі [70] автори розглядають особливості використання гнучких методів для розроблення екологічно чистого ПЗ [70], зосереджено увагу на факторах успіху насамперед його розробників. Вони вважають, що за останнє десятиліття використання гнучких методів для розроблення ПЗ різко зросло, позаяк гарантують швидку доставку зазвичай якісного ПЗ з підвищенням задоволення користувачів і зниженням вартості реалізації проекту. Однак, останніми роками через появу екологічної інженерії ПЗ його розробники змушені більше зосереджувати увагу на економічних і екологічних особливостях реалізації програмного проекту. Так звана "зелена" інженерія ПЗ спрямована на проектування, розроблення та використання ПЗ з обмеженими енергетичними та обчислювальними ресурсами. Автори роботи [70] стверджують, що інженери-програмісти, які професійно займаються глобальним розробленням ПЗ, адаптували гнучкі методи для швидкого, інтерактивного та безпечного його впровадження. У своєму дослідженні [70] автори визначили 16 факторів успіху за допомогою систематичного огляду літератури (англ. *Systematic Literature Review*, SLR) і застосували розроблені критерії пошуку, виведені на підставі різних проблем. Для цього було визначено та розглянуто 80 відповідних документів. Потім результати дослідження SLR були емпірично підтверджені за допомогою опитування в галузі розроблення ПЗ, в якому взяли участь 106 експертів із 25 різних країн. Результати такого промислового опитування респондентів здебільшого узгоджуються з результатами SLR. Однак, існує різниця в рангах різних факторів успіху в двох наборах даних – SLR та промислове дослідження (англ. *Industrial Survey*, IS).

У роботі [72] автори розглядають особливості реалізації розподіленого гнучкого розроблення ПЗ (англ. *Distributed Agile Software Development,* DASD) [3, 11, 52, 56, 77]. Вони вважають, що гнучке розроблення ПЗ – це нова стратегія, яка використовує гнучкі підходи для створення гнучкої архітектури ПЗ, яка може адаптуватися до мінливих потреб замовника. Все це призводить до постійно повторюваного поступового проектування гнучкого розроблення ПЗ. У своєму дослідженні [72] автори обговорюють поняття гнучкого розробленя ПЗ, його переваги, цілі та фактори, які впливають на нього, особливо на його функції. Після цього вони аналізують розподілені гнучкі команди розробників, їхні

переваги та загальні практики їхнього використання [66]. Автори стверджують [72], що в усьому процесі розроблення ПЗ Scrum-зустрічі відіграють важливу роль на всіх стадіях реалізації програмного проекту [30, 40, 61]. Це допомагає розробникам ПЗ ефективно організувати весь процес і, як наслідок, скорочує загальну тривалість реалізації проекту. Тому ввесь Scrum-процес [6, 2] на сьогодні також широко обговорюють, більше, ніж будь-коли раніше, і часто при цьому не помічають команди розробників, водночас як такі технології торкаються кожної стадії життєвого циклу програмного проекту [30, 40, 43, 61]. Автори [72] визначають поточну тенденцію, правила майбутнього розвитку та принципи створення складного ПЗ з метою вдосконалення розподіленого гнучкого розроблення ПЗ шляхом використання нових практик і досягнень в міру їх розвитку [66].

У роботі [56] автори розглядають особливості класифікації факторів ризику [58, 73, 2, 78] в розподіленому гнучкому розробленні ПЗ на підставі історії дій користувача. Вони вважають, що розподілене гнучке розроблення ПЗ (англ. *Distributed Agile Software Development*, DASD) [11, 52, 54, 56, 72, 77] – це модель життєвого циклу реалізації програмного проекту [30, 40], яку найчастіше використовують в ІТ-індустрії. Технологія DASD – це суміш розподіленого розроблення ПЗ (англ. *Distributed Software Development*, DSD) і гнучкого розроблення ПЗ (англ. *Agile Software Development*, ASD) [3, 11, 52, 56]. Незважаючи на те, що DASD об'єднує швидкість і економічні переваги ASD і DSD, ці методології привносять різні фактори ризику [73, 78], які виникають через їх протилежні, як на перший погляд, принципи роботи. Ці взаємопов'язані ризики необхідно вчасно виявити, проаналізувати та розробити заходи щодо їх локалізації та ліквідації з метою досягнення успіху виконання стадій та етапів реалізації проекту. У своєму дослідженні [56] автори вказують на важливість управління ризиками розроблення ПЗ в DASD. Вони аналізують наявну літературу та наводять фактори ризику [58, 2], пов'язані з DASD. Також автори [56] наводять поточні проблеми, висвітлені в наявній літературі, та пропонують нову техніку класифікації ризиків DASD на підставі історії дій користувачів. Вони вважають, що ця методика допоможе фахівцям-практикам визначити ризики [78], пов'язані з різними історіями дій користувачів [66]. Загалом їхнє дослідження представляє масштаби завдань вдосконалення процесу управління ризиками DASD, які допоможуть як практикам, так і дослідникам.

У роботі [52] автори наводять систематичний огляд інструментів і методів розподіленого гнучкого розроблення ПЗ [64, 70]. Вони вважають, що проект розроблення ПЗ – це певна методологія від збору вимог до тестування та підтримки, що завершується процедурами його впровадження в певні часові періоди для створення бажаного програмного продукту. Методологія Agile – це ітеративна методологія управління процесом реалізації проекту загалом і процесом розроблення ПЗ зокрема, яка дає змогу команді розробників ефективно надавати готові продукти проекту замовнику із мінімальними перешкодами. За гнучкого розроблення ПЗ планування проекту є надзвичайно важливою стадією, від ефективної реалізації якої зокрема залежить успіх реалізації програмного проекту загалом. Відсутність особистого спілкування розробників з замовниками та неможливість обміну паперовими індексними картками між усіма учасниками проекту мають значний негативний вплив на його планування в розподілених середовищах [67]. У своєму дослідженні [52] автори обговорюють переваги розподіленого гнучкого розроблення ПЗ та різні інструменти його планування, які доступні для успішного вирішення цих проблем. Ці інструменти відрізняються за функціями, діями, можливостями використання і вартістю як придбання, так і виготовлення. Робота [52] авторів мала на меті надати більш глибоке розуміння найсучасніших інструментів розподіленого гнучкого планування проекту, яке професіонали використовують повсякденно у своїй роботі. Гнучкі інструменти, які автори досліджували та порівнювали, є як відкритими, так і пропрієтарними (від англ. *Proprietary Software* – власницьке ПЗ). Вони також надають короткий опис 17 розподілених гнучких інструментів, таких як Jira, nTask, ActiveCollab, Monday.com, ProofHub, GitLab тощо. У роботі [52] автори також навели матеріали щодо повного вивчення цих інструментів з точки зору їхніх функцій, вартості придбання та використання.

У роботі [67] автори розглядають особливості тестування ПЗ в Agile середовищі. Вони стверджують, що гнучка методологія сприяє ітераційному та поетапному процесі розробленню ПЗ зі значним тестуванням його компонент. Ця методологія орієнтована на клієнта та розробника ПЗ, що успішно застосовує зміни вимог до нього під час процесу його розроблення. Незважаючи на те, що Agile-методології містять гнучкість у своїх процесах [3, 7], основний акцент автори зробили на задоволенні потреб замовників, які є ключовими компонентами технології реалізації програмного проекту. Однак, прийняти методологію Agile-розроблення ПЗ складно навіть передовим ІТ-компаніям [8, 11, 52, 55, 56, 59]. Щоб це запрацювало, потрібна правильна комбінація учасників проекту, насамперед кваліфікованої команди розробників, досвідчених менеджерів і замовників – знавців предметної області. Як наслідок, готовий продукт проекту виграє як від постійного тестування, так і активної участі клієнтів у такому тестуванні. У своєму дослідженні [67] автори обговорюють різноманітні методи реалізації Agile-проектів і наводять переваги тестування в Agile-середовищі. Вони вважають, що Agile-тестування є формою спільного тестування продуктів проекту як їх розробниками, так і безпосередніми користувачами. Користувачі беруть участь у визначенні приймальних тестів шляхом встановлення варіантів використання ПЗ та його атрибутів. Розробники ж співпрацюють із тестувальниками, щоб створювати тестові пакети, які можуть перевіряти автоматично функціональність ПЗ [15]. Гнучке тестування вимагає, щоб кожен учасник проекту був залучений до цього процесу, що вимагає багатьох раундів спілкування та творчої співпраці. Як і більшість особливостей Agile-розроблення ПЗ, Agile-тестування вимагає залучення безпосередніх користувачів якомога раніше протягом усього циклу реалізації програмного проекту.

У роботі [55] автори розглядають особливості управління якістю ПЗ за допомогою Agile тестування. Вони стверджують, що таке управління є критичним сегментом процесу розроблення ПЗ для інженерів з його якості. Це стає ще більш рутинним і складним завданням, коли процес розроблення ПЗ переходить на вико-

ристання гнучких інструментів і методів [52]. Оскільки методології та засоби, якими керує фреймворк гнучкого розроблення ПЗ (англ. *Agile Software Development*, ASD) [3, 52, 54, 56, 59, 77], є абсолютно відмінними від традиційних підходів забезпечення якості ПЗ та феноменального тестування [7], де присутня насамперед інтуїція, метапізнання та філософська методологія реалізації будь-якого проекту. Тому в методології ASD концентрація розроблення ПЗ зосереджена більш на людях, ніж на процесах, методах і засобах. Окрім цього, план тестування ASD має бути енергійним протягом усього процесу реалізації проекту, що забезпечує високоякісний його продукт завдяки ретельному відгуку замовників і їхнього аналізу розробниками. Плани тестування ASD потрібно писати після кожного етапу розроблення ПЗ чи випуску його поточних релізів, де тестування навантаження та продуктивності є одним із ключових елементів. Автори роботи [55] стверджують, що четвертий квадрант Agile-тестування (тестування навантаження та продуктивності) потрібно зосереджувати на нефункціональних вимогах, таких як продуктивність, безпека та стабільність. Інструмент для проведення навантажувального тестування з відкритим кодом Apache-JMeter можна використовувати для покращення якості ПЗ. Ця програма розроблена мовою Java, тому її часто використовують для аналізу продуктивності ПЗ, програм і веб-сторінок із функцією надійного інтегрованого середовища розроблення (англ. *Integrated Development Environment,* IDE) та запису плану тестування. Окрім цього, автори роботи [55] вважають, що четвертий квадрант достатньо здатний керувати методами тестування навантаження та продуктивності для статичних і динамічних ресурсів. Це стало можливим завдяки імітації тисяч користувачів цільового сервера або ПЗ-хосту та відповідей у формі статистики й графічних форматів.

Підсумовуючи зазначене вище, вважаємо, що гарантія якості програмного продукту є обов'язковою для будь-якого IT-бізнесу, незалежно від того, чи це продукт для виконання повсякденних операцій, чи для споживачів критичної інфраструктури. Завдання перевірки якості, викладені в стандарті IEEE 12207 для процесів життєвого циклу ПЗ, гарантують якість його розроблення та ефективність використання користувачами.

Результати дослідження та їх обговорення / Research results and their discussion

1. Стандарт IEEE 730-2014 і сфера його застосування. Гарантія якості ПЗ (англ. *Software Quality Assurance*, SQA) – засіб і практика моніторингу всіх процесів розроблення ПЗ, методів його виконання та робочих продуктів проекту для забезпечення їх відповідності визначеним завданням [10, 14, 16], а також може містити вказівки щодо дотримання настанов інших стандартів, таких як ISO/IEC/IEEE.

Стандарт IEEE 730-2014 [23] управляє SQA-діяльністю під час реалізації програмних проектів [19], його застосовують до розроблення ПЗ як частини системи, так і окремих його компонент. Цей стандарт стосується галузі виготовлення ПЗ, в т.ч. й проекти з розроблення нових або покращених наявних продуктів і їх обслуговування, а також проекти інтеграції ПЗ [10]. Зміст цього стандарту узгоджено з настановами, визначеними в інших стандартах, таких як ISO/IEC 12207:2018 [50] та ISO/IEC/IEEE 15289:2019 [51]. Стандарт IEEE 730-2014 не обмежений розміром, складністю, критичністю або застосуванням програмного продукту [13]. Він визначає дії, завдання та результати для SQA-процесів, проте не накладає строгих обмежень на впровадження чи виконання цих дій і завдань.

Стандарт IEEE 730-2014 визначає вимоги до видів SQA-діяльності [23], які можна виконувати протягом всього життєвого циклу ПЗ (рис. 1). В ньому визначено, що певним програмним проектам чи деяким IT-компаніям може не знадобитися використання всіх видів SQA-діяльності, зазначених у цьому стандарті [10]. Отже, його впровадження зазвичай передбачає вибір набору видів SQA-діяльності, які підходять для відповідної IT-компанії чи певних програмних проектів (рис. 2) [34].

Рис. 1. Життєвий цикл ПЗ / Software life cycle [22]

Використання стандарту IEEE 730-2014 має такі переваги [23]:

- зручний у використанні, надзвичайно інформативний:
  - легко слідувати його настановам, можна використовувати як довідник;
  - збирає всю поточну SQA-інформацію в одному місці;
  - надає чіткий контрольний перелік того, що потрібно зробити, щоб організувати виготовлення якісного ПЗ;
- виконує важливі для IT-компанії цілі гарантії якості продукту проекту:
  - демонстрація відповідності SQA-діяльності офіційному стандарту;
  - як орієнтир для розроблення ефективних і послідовних SQA-процесів, які мають відношення до IT-компанії;
  - отримання SQA-інформації та вказівок щодо конкретних питань.

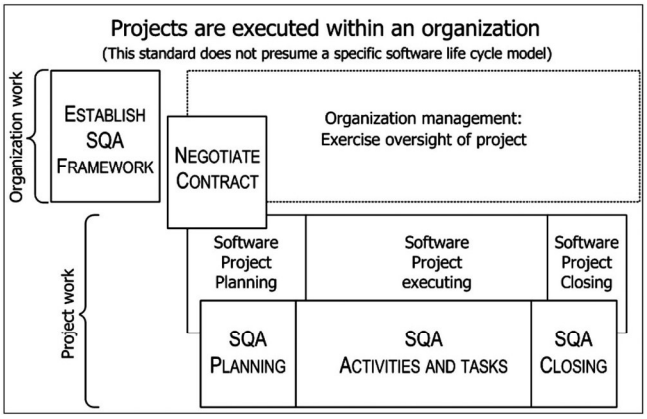

Рис. 2. Відносини між IT-компанією та програмним проектом / Relationships between the organization and the software project [23]

В будь-якій ІТ-компанії стандарт IEEE 730-2014 вигідно приймати до виконання таким працівникам [23]:

- менеджерам з якості ПЗ, яким потрібні вказівки та ефективне впровадження SQA-процесів;
- менеджерам програмних проектів, які не хочуть, щоб низька ефективність їх реалізації завдала шкоди бізнес-діяльності компанії;
- бюджету програмного проекту та здатності постачати замовнику якісне ПЗ;
- менеджерам продуктів проекту, які хочуть постачати ПЗ, яке повністю задовольняє потреби їхніх безпосередніх користувачів;
- керівникам вищої ланки, які хочуть, щоб ефективність бізнес-діяльності їхньої компанії була конкурентною перевагою на програмному ринку;
- керівництво ІТ-компанії, яке має замовників, що вимагають відданості затребуваній якості продуктів проекту;
- замовникам і безпосереднім користувачам, яким потрібне якісне ПЗ з невеликою кількістю помилок або взагалі без них, якщо таке може виконати його розробник.

Гарантія якості ПЗ (англ. *Software Quality Assurance*, SQA) – методологія перевірки відповідності процесів розроблення ПЗ попередньо зазначеному набору стандартів [42]. Забезпечення якості ПЗ відбувається до, під час і після життєвого циклу реалізації проекту (рис. 3). ІТ-компанії мають переконатися в тому, що внутрішні та зовнішні характеристики програмного продукту відповідають вимогам видів SQA-діяльності. Зовнішні якості ПЗ описують, як воно працює в режимі реального часу, тоді як внутрішні якості стосуються більш фундаментальних складових блоків ПЗ, таких як архітектура, структура бази даних чи програмний код. Прикладами зовнішньої якості ПЗ є його надійність, ефективність, вартість обслуговування та ремонтопридатність [10]. Внутрішня якість ПЗ стосується його складності, структури, гнучкості, читабельності, тестування та методів кодування, які використовують під час реалізації програмного проекту.

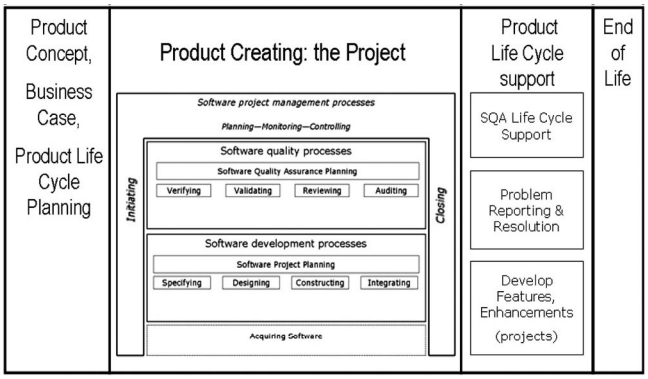

Рис. 3. Приклад життєвого циклу продукту, контексту для програмного проекту ПЗ / Example product life cycle, context for a software project [23]

Існує два основних підходи до забезпечення якості ПЗ: підхід до управління дефектами; підхід до атрибутів якості ПЗ.

Підхід до управління дефектами ПЗ реалізують шляхом підрахунку дефектів і за рахунок управління процесом їх усунення. Дефекти охоплюють широкий діапазон помилок від погано оброблених даних до недосконалого програмного коду [13]. В практиці управління дефектами, коли команда проекту їх виявляє, вони призначають їм категорії складності відповідно до серйозності дефектів. Потім команда вживає конкретних заходів для усунення виявлених дефектів. Зазвичай, такий підхід найкраще реалізують з чіткими та лаконічними контрольними діаграмами для вимірювання та покращення можливостей процесу розроблення ПЗ [14]. Його здійсненність визначає, наскільки наявні виробничі процеси відповідають настановам прийнятих стандартів.

Підхід до атрибутів якості ПЗ зосереджує увагу на дотриманні деяких характеристик його якості. Залежно від наявної ситуації, існує від шести до десяти й більше цих характеристик. Причина в тому, що деякі атрибути якості ПЗ можуть накладатися один на іншого відповідати одному з них. Наприклад, придатність ПЗ залежить від його функціональності, водночас як зручність використання розширює можливості навчання. Тим не менше, тільки ці шість основних атрибутів охоплять потрібну інформацію на даний момент – його функціональність. Тому виконання такого функціоналу потребує згодом менших ресурсів на його обслуговування, оскільки цей продукт загалом відповідає надійності, зручності використання, ефективності, ремонтопридатності та портативності.

Важливість гарантії якості ПЗ. Не тільки заради безпосереднього користувача, але й для самого програмного продукту важливо, щоб компанії-виробники використовували надійний SQA-процес для встановлення базового рівня очікувань щодо якості продукту [16]. Тому настанова стандартів щодо забезпечення якості ПЗ є надійним засобом високоякісного його виготовлення. Зазвичай, замовник завжди хоче мати якісне ПЗ, інакше він має мати вагомі причини, чому для нього є необов'язковим покращення якості ПЗ [15].

Насамперед, якісний програмний продукт заощаджує час і гроші безпосередніх його розробників. Однією з головних цілей будь-якої ІТ-компанії є надання потенційному користувачу продукту, який варто купити. Якщо йдеться про ПЗ, то гарантія його якості може підтвердити, що цей програмний продукт вартий того, щоб його придбати [16, 10, 21].

Проблема з програмним продуктом, який не варто купувати, полягає в тому, що команда проекту витрачає час і гроші, розробляючи його, і не отримує жодної копійки прибутку, щоб бути достойним конкурентом на ринку ПЗ. У випадку, якщо такий продукт все ж таки хтось і придбає, то його розробник буде витрачати найбільше часу та нести значні фінансові витрати на його обслуговування, якщо цей продукт будуть ще й використовувати [19]. Загалом гарантія якості ПЗ залишається ключовим фактором у масштабуванні бізнес-діяльності ІТ-компанії, а також у збереженні достойної її репутації й, щонайважливіше, бренду.

Важливість плану забезпечення якості ПЗ. Знак відповідності – захищений в установленому порядку символ, який свідчить, що маркований ним програмний продукт відповідає конкретному стандарту чи іншому нормативному документу [21, 10]. Маркування деякого продукту цим знаком здійснює орган із сертифікації, що видав знак відповідності, або компанія-виробник, якщо вона має на це ліцензію. Переваги компанії-виробника, якій надано сертифікат якості програмного продукту, є такими:

- піднімає довіру до якості продукту, який можна експортувати в промислово розвинені країни;
- розширює ринок збуту продуктів проекту;
- забезпечує продукти проекту рекламою і, як наслідок, збільшенням обсягів їх виготовлення (продажу);
- позбавляє конкуренції з боку виробників не сертифікованих програмних продуктів.

Переваги, які дає сертифікація програмного продукту його споживачам:

- захищає від програмних продукті, небезпечних для життя й здоров'я людини, а також її майна;
- полегшує вибір затребуваного програмного продукту;
- опосередковано сприяє підвищенню якості програмного продукту;
- безпосередньо покращує якість виконуваних робіт.

Знак відповідності засвідчує дотримання програмним продуктом вимог (рис. 4), які було встановлено під час виконання сертифікаційних робіт [20]. Такі знаки можуть нести різну інформацію про ІТ-компанію, яка виготовляє різні програмні продукти, у т.ч. й про:

- небезпеку конкретного продукту для споживача;
- вплив продукту на навколишнє середовище;
- довговічність продукту на ринку ПЗ;
- відповідність робочих характеристик продукту проекту нормативним;
- приналежність знаку відповідності конкретному постачальнику або виробнику;
- умови праці компанії-виробника та наявність в ній системи управління якістю ПЗ.

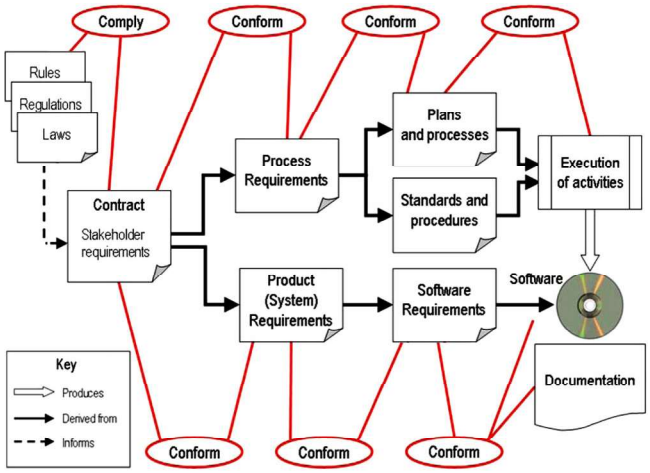

Рис. 4. Зв'язки для визначення відповідності між артефактами програмного проекту / Relationships for getermining conformance between software project artifacts [23]

План забезпечення якості ПЗ (англ. *Software Quality Assurance Plan*, SQAP) містить процедури, методи та інструменти, які використовують для підтримки відповідності програмного продукту потребам замовника (рис. 5), визначених у специфікації вимог до ПЗ (англ. *Specification of Software Requirements*, SRS) [19].

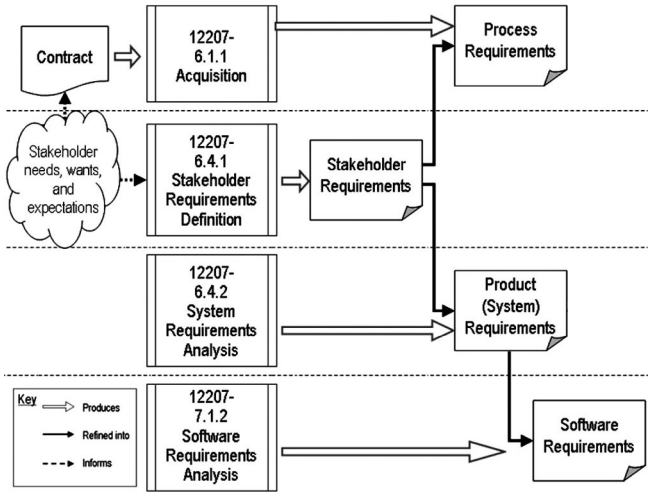

Рис. 5. Виведення вимог і взаємозв'язків між рівнями вимог / Derivation of requirements and relationships between levels of requirements [23]

План забезпечення якості визначає обов'язки SQA-команди розробників ПЗ та перераховує сфери, які необхідно переглянути та перевірити. Він також визначає робочі продукти SQA [10, 17]. Документ SQA-плану має містити такі розділи:

- розділ призначення;
- розділ довідкової інформації;
- розділ управління конфігурацією ПЗ;
- розділ повідомлення про проблеми та їх коригування;
- розділ використовуваних інструментів, технологій та методологій;
- розділ контролю програмного коду;
- розділ збирання, обслуговування та зберігання;
- методологію тестування.

Нижче наведено перелік SQA-діяльності [18]:

1) *Створення плану управління SQA*. Основна SQA-діяльність містить протоколи складання правильного плану щодо того, як SQA буде виконувати у поточному проекті. Окрім цього, якого б підходу SQA-команда розробників дотримувалася, які інженерні б заходи виконувала, все це також міститиме забезпечення правильного поєднання їхніх талантів під час розроблення ПЗ.
2) *Встановлення контрольних точок*. SQA-команда встановлює різні контрольні точки, згідно з якими вона оцінює ефективність проектної діяльності на кожній контрольній точці/етапі реалізації проекту. Це забезпечує регулярну перевірку якості ПЗ та ефективність роботи команди за графіком.
3) *Застосування методів розроблення ПЗ* допомагає його команді проекту досягти досконалих специфікацій вимог до нього. Для збору інформації аналітик може використовувати такі методи, як інтерв'ю (англ. *Interview*) та системну техніку функціонального аналізу (англ. *Functional Analysis System Technique*, FAST). Пізніше, на підставі зібраної інформації, керівник проекту може його оцінювати за допомогою таких методів, як структура розподілу робіт (англ. *Work Breakdown Structure*, WBS), кількість рядків коду (англ. *Source Lines of Code*, SLOC) і аналіз функціональних точок (англ. *Conversion Functional Point*, CFP).
4) *Виконання формальних технічних перевірок* (англ. *Formal Technical Inspection*, FTI) проводять для оцінювання ефективності реалізації проекту та досконалості архітектури прототипу продукту проекту. У цьому процесі проводять нараду з технічним персоналом для обговорення фактичних вимог до якості майбутнього ПЗ та якості його прототипу. Ця SQA-діяльність допомагає виявляти помилки на ранній стадії його життєвого циклу (англ. *Software Development Lifecycle*, SDLC) і зменшує зусилля для його перероблення на пізніших стадіях проекту (рис. 6) [74].

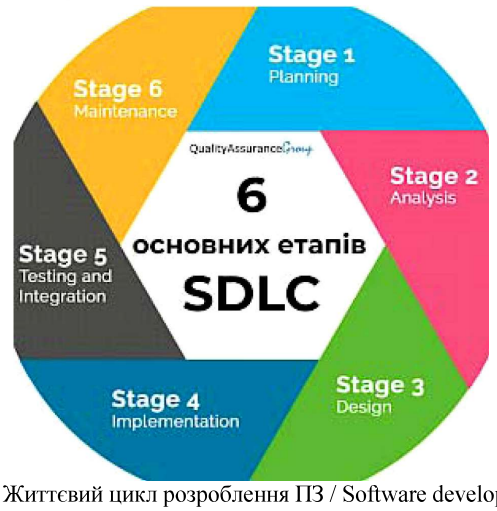

Рис. 6. Життєвий цикл розроблення ПЗ / Software development lifecycle [74]

5) *Наявність стратегії мультитестування*, під яким розуміють, що не варто покладатися на якийсь окремий підхід до тестування продуктів проекту, натомість потрібно проводити декілька типів тестування, щоб програмний продукт можна було добре перевірити з усіх точок зору для забезпечення кращої його якості.
6) *Забезпечення дотримання процесу*. Ця SQA-діяльність наполягає на потребі дотримання SQA-процесів під час розроблення ПЗ.

Процес розроблення ПЗ також має дотримуватися визначених процедур, а саме:

1) *Оцінювання програмного продукту*. Ця SQA-діяльність підтверджує, що розроблений продукт відповідає вимогам, виявленим у плані управління проектом. Загалом це забезпечує правильне дотримання прийнятих стандартів під час реалізації проекту.
2) *Моніторинг процесу* розроблення ПЗ. Ця SQA-діяльність перевіряє, чи були запроваджені правильні етапи для розроблення якісного ПЗ. Це роблять шляхом зіставлення фактично вжитих етапів із задокументованими.
3) *Контроль внесення змін*. У цій SQA-діяльності команда проекту використовує поєднання ручних процедур і автоматизованих інструментів, щоб мати механізм контролю за внесенням змін. Перевіряючи запити на зміни, оцінюючи їхній характер і контролюючи їх вплив, у такий спосіб забезпечують збереження якості ПЗ на етапах його розроблення та обслуговування.
4) *Вимірювання впливу змін на хід реалізації проекту*. Якщо група контролю якості ПЗ повідомляє про будь-який дефект у ньому, тоді відповідна команда проекту його виправляє. Після цього група контролю повинна визначити вплив змін на функціонал ПЗ, спричинених виправленням цього дефекту. Їм потрібно перевірити не тільки те, чи зміни усунули дефект, але й чи сумісні зміни з усім функціоналом ПЗ. Для цього команда проекту використовує показники якості ПЗ, які дають можливість менеджерам спостерігати за їхніми діями та запропонованими змінами від початку до кінця SDLC, а також ініціювати коригувальні дії, якщо це необхідно.
5) *Виконання SQA-аудиту* здійснюють для перевірки всього фактичного SDLC-процесу, а потім порівнюють його з нормативним процесом. Аудит також перевіряє, чи все, про що повідомила команда проекту у звітах про статус компонент ПЗ, було насправді виконано чи ні. Ця SQA-діяльність також виявляє будь-які проблеми, спричинені їх невідповідністю, і дає рекомендації щодо їх ефективного усунення.
6) *Ведення записів і звітів*. Вкрай важливо зберігати необхідну документацію щодо SQA-діяльності та ділитися із зацікавленими сторонами необхідною SQA-інформацією. Результати випробувань та аудиту, звіти про перевірку, документацію із запитами на внесення змін тощо варто зберігати для використання в майбутньому програмному проекті.

Методи SQA. Існує декілька SQA-технік. Аудит є основною технікою, яка широко поширена в багатьох IT-компаніях. Однак відомі декілька інших важливих технік [25], різні методи яких містять:

1) *Аудит* – передбачає перевірку робочих продуктів і пов'язаної з ними інформації, щоб визначити, чи дотримувався проект набору стандартних процесів чи ні.
2) *Рецензування*. Нарада, на якій програмний продукт розглядають як внутрішні, так і зовнішні зацікавлені сторони, щоб отримати їхні коментарі та схвалення.
3) *Перевірка коду*. Це найбільш офіційний вид перевірки, який виконує статичне тестування, щоб знайти помилки та уникнути зростання кількості дефектів на пізніших етапах реалізації проекту. Це виконує навчений посередник – рецензент і базується на правилах, контрольному списку, критеріях входу та виходу. Рецензент не повинен бути автором програмного коду.
4) *Перевірка архітектури* – виконують за допомогою контрольного списку, який вказує на такі області розроблення ПЗ: загальні вимоги до ПЗ та оформлення інтерфейсів користувача; функціональні та інтерфейсні характеристики системи; конвенції; відстеження вимог; програмні структури та відповідні інтерфейси; логіка; продуктивність; оброблення помилок і відновлення; тестованість, розширюваність; зчеплення та когезія.
5) *Моделювання* – інструмент, який імітує реальну ситуацію з метою віртуального дослідження поведінки програмних компонент у реальній системі.
6) *Функціональне тестування* – метод контролю якості ПЗ, який перевіряє, що робить програмна система, не розглядаючи, як саме вона це робить. Цей тип тестування "чорної скриньки" в основному зосереджений на перевірці специфікацій вимог або функцій системи.
7) *Стандартизація* відіграє вирішальну роль у забезпеченні якості ПЗ, зменшує неоднозначність і припущення, гарантуючи в такий спосіб його якість.
8) *Статичний аналіз* виконують автоматизованим інструментом без фактичного виконання програми. Цей метод широко використовують для забезпечення якості ПЗ критичних інфраструктур: медичних, ядерних і авіаційних. Метрики ПЗ та зворотне проектування є популярними формами статичного аналізу.
9) *Поетапні настанови щодо розроблення ПЗ чи його програмного коду* – різновид експертної перевірки, коли рецензент вимагає від членів команди проекту пройтись продуктом і поставити запити, запропонувати альтернативи та зробити коментарі щодо можливих помилок, стандартних порушень протоколу виконання роботи або будь-яких інших завдань.
10) *Тестування шляху* – метод тестування "білої скриньки", де повне покриття гілок забезпечують виконанням кожного незалежного шляху принаймні один раз.
11) *Стрес-тестування* проводять, щоб перевірити, наскільки надійна система, випробовуючи її під великим навантаженням, тобто поза нормальними умовами її функціонування.

Отже, гарантія якості програмного продукту є обов'язковою для будь-якого IT-бізнесу, незалежно від того, чи це продукт для виконання повсякденних дій, чи для споживачів критичної інфраструктури. Стандарт IEEE 730-2014 надає вказівки та встановлює вимоги до якості ПЗ під час реалізації програмних проектів [23], а також надає настанови щодо ефективного його використання. Він також описує SQA-процеси, тобто комплексну діяльність IT-компанії, яку вони мають використовувати протягом усього життєвого циклу ПЗ.

Існують різні види діяльності IT-компанії, стандарти та методи, яких вони мають дотримуватися, щоб гарантувати належну якість ПЗ і відповідати потребам замовника. Встановлено, що SQA-діяльність скорочує витрати на розроблення ПЗ, які інакше б виникли, якщо воно згодом повернеться команді проекту для усунення дефектів. При цьому буде витрачено значно більше ресурсів компанії, намагаючись виявити причини їх виникнення, а також усунути, враховуючи зв'язки між вимогами. Для замовника також проблематично призупинити свою роботу через дефекти ПЗ, що може призвести до втрати його довіри його розробника та зниження шансів на довготермінові взаємовідносини.

2. Впровадження SQA-процесів у проект розроблення веб-додатку "Інтернет-банкінг". Ефективні SQA-процеси визначають дії, які потрібно виконувати і їх підтверджувати, що вони виконані, вимірювати та відстежувати відповідні процеси, розробляти заходи для управління ними та їх вдосконалювати, а також за-

охочувати команду проекту використовувати ці процеси для виготовлення програмних продуктів, які відповідають встановленим вимогам [26]. Водночас, SQA-процеси вимагають свого постійного вдосконалення на підставі об'єктивних вимірювань і фактичних результатів реалізації проектів [28].

Визначення процесів забезпечення якості продукту проекту. Згідно з настановами стандарту ISO/IEC 12207:2018 [50], SQA-діяльність має дати команді проекту такі результати його реалізації:

1) Визначити роль SQA-функції в ході реалізації проекту.
2) Встановити організаційні процеси, які не залежать від SQA-процесів.
3) Встановити організаційну політику IT-компанії, яка визначає та управляє ролями, а також обов'язками SQA на проекті.
4) Встановити спосіб нагляду за виконанням SQA-діяльності, її завдань і результатів разом із способом надання зворотного зв'язку SQA-функції.
5) Встановити спосіб аналізу досвіду виконання SQA-функцій під час реалізації поточних і попередніх проектів, а також ділитися отриманими знаннями з іншими проектами.
6) Призначити відповідальних за виконання SQA-ролей у IT-компанії, а також під час реалізації конкретного проекту, який є організаційно незалежним як від управління ним, так і від методології розроблення ПЗ.

Для досягнення цілей реалізації проекту потрібно запровадити координацію та верифікацію продукту проекту, його підтвердження та перегляд, аудит та інші настанови стандарту ISO/IEC 12207:2018 [50]. Оскільки SQA-функція співпрацює з управлінням проекту, то потрібно з'ясувати, які з інших 39 процесів, визначених у цьому стандарті, варто координувати з SQA-діяльністю.

Отже, виходячи з зазначених вище потреб, можна сформувати структуру SQA-процесів для продукту проекту "Інтернет-Банкінг", який було протестовано в ході проведення цього дослідження.

Інтернет-банкінг (англ. *Internet Banking*) або веб-банкінг – один із видів дистанційного банківського обслуговування клієнтів, засобами якого забезпечують доступ до рахунків і операцій за ними в будь-який час та з довільного комп'ютера через мережу Інтернет [42]. Для виконання операцій використовують стандартний браузер (Google Chrome, Internet Explorer, Opera, Mozilla тощо). Отже, потреби встановлювати додаткове ПЗ немає [34].

Інтернет-банкінг має містити такі послуги:

- інформацію про картки клієнта;
- блокування картки клієнтом у разі її викрадення або втрати;
- виписки за рахунками;
- інформацію про інші відкриті банківські продукти (платіжні картки, депозити, кредити, інше);
- платежі в межах банку;
- платежі в національній валюті в межах країни;
- оформлення заяв на під'єднання до інших послуг (sms-банкінг, картки, депозити, кредити, інше).

Інтернет-банкінг має містити такі додаткові послуги для задоволення потреб клієнта:

- встановлення лімітів на різні види операцій (оплата через мережу Інтернет, термінал, банкомат і т. д.) з карткових і поточних рахунків, наприклад 0 (нуль);
- платежі в іноземній валюті;
- обмін валют;
- оплату рахунків про надані небанківські послуги (зокрема, комунальні, зв'язок);
- придбання ваучерів передплачених послуг (мобільні оператори, мережа Інтернет).
- пряме поповнення балансу SIM (USIM, R-UIM)-карти за вказаним номером телефону українських мобільних операторів.
- поповнення Skype-рахунку.

Це ті основні вимоги замовника до наданих послуг інтернет-банкінгом, які потрібно було протестувати в проекті його розроблення.

Етапи забезпечення якості продукту проекту (рис. 7). Розроблення інтернет-банкінгу містить такі процеси [35]: аналіз вимог; архітектуру ПЗ (дизайн); реалізацію; перевірку або тестування; технічне обслуговування. Для забезпечення якості веб-продукту потрібно ввести такі SQA-процеси: аналіз вимог; планування та написання тестів; модульне та інтеграційне тестування; тестування системи та її продуктивності; тестування безпеки; кросбраузерне / кросплатформне тестування; оновлення тестів; регресійне тестування.

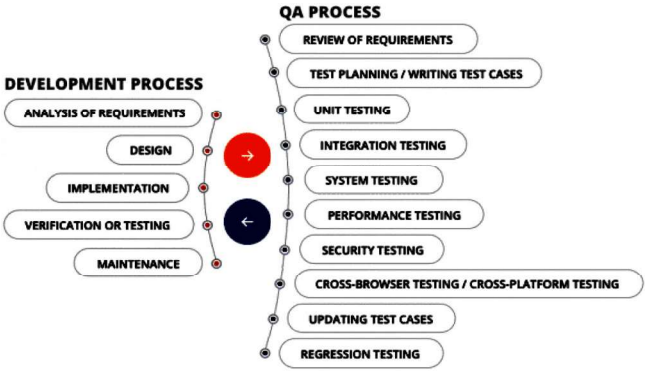

Рис. 7. Процеси розроблення ПЗ і забезпечення його якості / Software development and quality assurance processes [39]

Етап перший: ознайомлення з вимогами та документацією до ПЗ [36]. Інженери з QA (англ. *Quality Assurance* – забезпечення якості) починають роботу над проектом паралельно з формуванням відповідної документації [25, 26]. Вони розглядають вимоги та наявну документацію до продукту проекту "Інтернет-банкінг" на відповідність таким критеріям: повнота; скорочення штатів; зрозумілість; послідовність; виконуваність; можливість перевірки.

Прикладом вимоги, яка відповідає зазначеним вище критеріям, є формат, який задає питання: Хто користувач? Яка потреба в користувача? Який результат? У потенційного користувача такого програмного продукту мало б виникнути таке побажання: "Я, як інженер з підтримки клієнтів кредитного сектору, хочу мати окремий фільтр на адмін-панелі для того, щоб міг виключно відфільтровувати запити клієнтів, які стосуються їхніх кредитних питань".

Для перетворення такого побажання у користувацькі вимоги до майбутнього ПЗ потрібно виконати відповідну декомпозицію відповідей на питання:

1) Хто користувач – інженер з підтримки клієнтів кредитного сектору.
2) Яка потреба в користувача – мати окремий фільтр на його робочій панелі.
3) Який результат має приносити така дія – фільтрувати запити клієнтів кредитного сектору.

Отже, мета таких дій полягає в аналізі архітектури системи та технологій її реалізації на наявність розбіжностей. Ключові переваги таких дій для процесу розроблення веб-додатку є такими:

- помилки коштують менше, якщо їх виявляють на ранніх етапах реалізації проекту;
- якісно підготовлена документація означає більш належну якість продукту проекту з меншими затратами праці та більш точними оцінками його складових.

Для успішної реалізації цього етапу використовують спеціалізоване ПЗ для перегляду та виправлення документації, наприклад Confluence – простір для командної роботи, зручний для розподілених команд його виконання. Передбачено, що нагромаджені командою знання мають бути поєднані із можливостями програмної системи для спільної роботи. Тут можна зібрати всю відповідну документацію, яку використовують протягом певного періоду реалізації проекту, що дає змогу підтримувати внутрішню базу знань, яку можна використати як у цьому, так і в майбутніх проектах. Кожен член команди проекту може бачити будь-які зміни, щойно яка не-будь вимога чи документ поміняють свій статус, а за потреби їх можна додати, оновити чи видалити.

Етап другий: планування та підготовка тестових випадків [33]. Коли вимоги до ПЗ узгоджено з його замовником, а також настає час почати планувати тестові випадки, тобто – описувати дії, які мають виконувати інженери з групи контролю якості ПЗ, щоб переконатися, що його частина функціонує відповідно до плану. Якщо кількість таких конкретних випадків виявиться дійсно значною, то будь-хто з тестувальників може скористатися спеціальними інструментами, наприклад TestRail або Zephyr для написання відповідних тестів. Обидві частини ПЗ дають можливість створювати та модифікувати тести та відстежувати результати за допомогою певних показників.

Етап третій: розроблення тестових випадків [33]. Після завершення етапу розроблення ПЗ група контролю його якості починає запускати тестові випадки. Основна мета цього етапу – перевірити, чи правильно прийнято рішення з технічної точки зору та чи відповідає початковим вимогам власника продукту проекту.

Табл. 1. Приклад тестового випадку / An example of a test case

ID	Модуль	Опис	Платформа	Статус
1	Платіжна система	Потрібно перевірити, чи клієнт може здійснити переказ коштів з однієї картки на іншу	Андроїд	Пройдено

Інженери з групи контролю якості ПЗ мають такі основні види діяльності та відповідні цілі [13]:

1) *Випробування на дим*, де ці інженери ретельно перевіряють, чи воно або його модуль функціонує за планом. Після впровадження згаданих дій починають подальше розслідування.
2) *Інтеграційне тестування*, де здійснюють перевірку, щоб різні компоненти працювали як єдина система.
3) *Тестування продуктивності* поділяють на тестування навантаження та стрес-тестування:
   - *тестування навантаження* – здійснює перевірку поведінки системи для нормального та очікуваного пікового навантаження;
   - *стрес-тестування* – визначає критичне навантаження, після якого система виходить з ладу;
4) *Тестування безпеки*, де переконуються в тому, що вебдодаток має достатній рівень захисту.
5) *Кросбраузерне / кросплатформне тестування*, де здійснюють перевірку безперебійної роботи програми в різних браузерах (Chrome, Mozilla, Safari) або на різних платформах (Android, iOS, Windows Phone). Це особливо важливо для веб- та гібридних програм [36, 33].
6) *Регресійне тестування*, де виявляють помилки у попередньо перевіреному коді. Зазвичай, це потрібно під час додавання нових функцій або оновлення наявної системи. Знову ж таки, тестувальник може вибрати ручне чи автоматизоване тестування (наприклад, модульне чи регресійне). Тут функціонує таке загальне правило: *чим довше триває реалізація проекту, тим важливіше має значення автоматизоване тестування*.

Тестування безпеки є критично важливим заходом для банківських додатків. Під час підготовки до тесту група контролю якості ПЗ повинна передбачити як негативні, так і позитивні сценарії тестування, щоб зламати систему та повідомити про її наслідки до того, як будь-який неавторизований користувач-зловмисник отримає до неї доступ. Щоб запобігти зламу системи, банк також має запровадити багаторівневу перевірку доступу до неї, наприклад, надання одноразового паролю. Для тестування безпеки веб-додатку використовують інструменти автоматизації, такі як IBM AppScan і HPWebInspect, а для ручного тестування – такі інструменти, як Proxy Sniffer, Paros proxy, HTTP watch тощо.

Етап четвертий: звіт і вимірювання [28, 30]. Коли інженер із забезпечення якості ПЗ виявляє в ньому помилку, то після цього він записує її в систему відстеження помилок, яка водночас є системою управління проектом. Для цього інженер може використовувати такі інструменти, як Jira або Redmine, кожен з яких має широкі можливості налаштування. Вони дають змогу легко відстежувати проблеми будь-якого рівня, від несправної форми входу в систему до проблем її безпеки, і всі члени команди проекту можуть бачити оновлення чергових завдань у режимі реального часу. Це спрощує спілкування всередині команди проекту та допомагає мати чіткий огляд процесу вдосконалення як технології розроблення веб-додатку, так і його самого.

Кожна проблема отримує рівень пріоритету її вирішення від термінового до низького, яку потім реалізує хтось з команди проекту залежно від часу та ступеню їх зайнятості.

Табл. 2. Приклад виявлення помилки / An example of error detection

ID	Загальна назва	Умови виникнення	Етапи відтворення	Актуальний результат	Очікуваний результат
1	Після здійснення платежу баланс клієнта не оновився	iPhone, версія операційної системи 11	Увійти в аккаунт клієнта. Здійснити переказ коштів з однієї картки на іншу. Перевірити баланс.	Після здійснення платежу баланс клієнта не змінився	Баланс клієнта оновився відповідно до здійсненого платежу

Етап п'ятий: перевірка виправлень. Коли хтось з команди проекту вирішує проблему, то він інформує відповідальних інженерів з якості ПЗ, які перевіряють якість її вирішення. Заяву в системі відстеження помилок закривають, якщо не виявлено жодного застереження. Тут застосовують таке правило: *жодну помилку не можна позначити як виправлену, доки її не буде перевірено*.

3. Розгортання ПЗ для безпосереднього користувача. Після виконання всіх зазначених вище SQA-процесів стосовно забезпечення якості продукт проекту "Інтернет-банкінг" його можна представити безпосередньому користувачу [18, 20] для його тестування в реальних виробничих умовах. Розгортання ПЗ містить дії, необхідні для того, щоб зробити програмну систему доступною для відповідних користувачів.

Сьогодні більшість ІТ-компаній розгортають розроблене чи оновлене ПЗ, а також виправлене за допомогою поєднання ручних і автоматизованих дій. Деякі з найпоширеніших дій із розгортання ПЗ містять його підготовку, встановлення, тестування, контрольну перевірку та моніторинг продуктивності. Розгортання ПЗ є одним із найважливіших етапів процесу його розроблення. Це механізм, за допомогою якого програму, її модулі, оновлення чи виправлення доставляють від розробника до безпосередніх користувачів. Спроби, що використовують члени команди проекту для створення, тестування та розгортання програмного коду, впливають на те, наскільки швидко готовий продукт буде реагувати на зміни в уподобаннях або вимогах його безпосередніх користувачів, а також на якість кожної зміни.

Команда проекту, яка виконує процес розроблення ПЗ, реалізує його тестування та розгортання, може швидше реагувати на попит безпосередніх користувачів за допомогою нових версій і частішого надавання нові функції, щоб підвищити їх задоволеність і скористатися фінансовими можливостями замовника проекту.

Адже відомо [58], що протягом останніх двох десятиліть команди розробників ПЗ впроваджували значні інновації, створюючи нові парадигми та методи роботи щодо доставки замовнику якісного продукту, розробленого для задоволення його мінливих потреб. Зокрема, розробники ПЗ створили робочі процеси, які дають можливість швидше та частіше розгортати його нові версії в робочому середовищі замовника, де до нього можуть отримати доступ користувачі в будь-який час і з різних місць, за умови вільного доступу до мережі.

Процес розгортання ПЗ стосується його інсталяції на сервері замовника чи на будь-якому його пристрої. Оновлене ПЗ або будь-яку його версію можна розгорнути на тестовому сервері чи машині замовника або в живому середовищі. ПЗ можна розгортати декілька разів протягом періоду його розроблення, щоб перевірити належне функціонування його компонент на наявність помилок. Іншим прикладом розгортання ПЗ може бути ситуація, коли користувач завантажує мобільну програму з Integration Store і встановлює її на свій мобільний пристрій.

Отже, розроблення ПЗ – конкретна версія програмного коду та його залежностей, доступна для розгортання в середовищі замовника. Розгортання ПЗ стосується процесу його роботи на цільовому пристрої замовника, будь то тестовий сервер, робоче середовище, комп'ютер або мобільний пристрій користувача.

Існують різні типи стратегій розгортання ПЗ: базовий; мультисервісність; прокатка; Blue Green Deployment; Canary; A/B тестування.

Хоча багато команд проекту все ще обирають розміщення веб-додатків за допомогою локальної ІТ-інфраструктури, постачальники хмарних послуг, як-от Amazon Web Services (AWS), Google Cloud Platform і Microsoft Azure, пропонують інфраструктуру як послугу (англ. *Infrastructure as a Service*, IaaS) і платформу як послугу (англ. *Platform as a Service*, PaaS). Це продукти, які допомагають командам проектів розгортати свої програми в таких середовищах без додаткового фінансового та адміністративного навантаження, пов'язаного з управлінням власним сховищем даних і серверами віртуалізації [34].

Особливості розгортання банківського додатку. Всі банківські додатки використовують розгортання типу Canary (канарка). Canary-реліз – стратегія розгортання продукту, в межах якої зміни вимог спочатку стосуються невеликої групи користувачів. Такий підхід передбачає поетапне розгортання продукту, спочатку для невеликої групи користувачів, щоб вони могли протестувати його та надіслати свої відгуки – претензії, зауваги та пропозиції [28]. Щойно зміна вимог буде прийнята, оновлене ПЗ стане доступним для решти користувачів. Розгортання типу Canary показують розробнику, як користувачі взаємодіють зі змінами програми в реальному середовищі. Як і в "синьо-зелених" (англ. *Blue Green Deployment*) розгортання, стратегія Canary пропонує оновлене ПЗ без простоїв і легких відкотів. На відміну від "синьо-зеленого", канаркові розгортання більш плавні, а збої роботи мають обмежений вплив на всіх її користувачів. Це зумовлено тим, що в разі виникнення в користувачів певних багів, непередбачуваних помилок, збоїв, витоків конфіденційної інформації тощо, процес роботи ПЗ можна легко згорнути до попереднього стану, і кількість клієнтів, які зазнали негативного впливу, буде обмежена. Нагадаємо, що під баґами (англ. bug – жук, комашка) розуміють помилку, ваду або дефект в комп'ютерній програмі чи системі, що викликає в ній неправильний або неочікуваний результат чи неочікувану поведінку. Термін використовують стосовно означення помилок, котрі виявляють на стадії роботи програми, на відміну від помилок проектування чи синтаксичних помилок. Зазвичай баґи локалізують та виправляють у процесі тестування та доробки програми.

Отже, тестування ПЗ (англ. *Software Testing*) з використанням розглянутих в роботі настанов стандартів – процес технічного дослідження, призначений для виявлення інформації про якість продукту проекту відносно контексту, в якому його мають використовувати. Використана техніка тестування містила як процес пошуку помилок або інших дефектів, так і випробування програмних складових для оцінювання їх функціональності. Це процес перевірки відповідності заявлених до продукту користувацьких вимог і реально реалізованої його функціональності, яку здійснюють шляхом спостереження за його роботою в штучно створених ситуаціях і на обмеженому наборі тестів, спеціально обраних.

Внаслідок виконаного тестування веб-додатку "Інтернет-банкінг" було отримано таке оцінювання:

- відповідність системним вимогам, якими керувалися його проектанти та розробники;
- правильність відповіді запитам користувача для всіх можливих вхідних даних;
- виконання затребуваних функцій замовника за прийнятний час;
- практичність роботи користувачів з його інтерфейсами;
- сумісність із іншим ПЗ та операційними системами;
- відповідність користувацьким потребам замовника.

Також з'ясовано, що банківська сфера є найбільш вразливою для кібератак, тому захист ПЗ вимагає ретельного тестування. Чітке уявлення того, що потрібно для тестування банківського домену та наскільки це важливо для замовника, слугує успішній як реалізації продукту проекту, так і його безпосередньому використанню замовником. Тут треба розуміти, що:

- більшість банківського ПЗ розроблено на мейнфреймах і Unix;
- тестування ПЗ допомагає зменшити можливі збої в його роботі під час безпосереднього використання;

- належне тестування ПЗ та дотримання галузевих стандартів позбавляють ІТ-компанії штрафних санкцій;
- хороші практики допомагають досягти ефективних результатів, репутації та більшого бізнесу для ІТ-компаній;
- як ручне, так і автоматичне тестування ПЗ мають відповідні переваги та зручність використання.

Обговорення результатів дослідження. Різні методи розроблення ПЗ вимагають постійного вдосконалення, однак провідні ІТ-компанії визнають, що все ще існують невирішені проблеми реалізації програмних проектів, у т. ч. й у гнучкому середовищі [37, 67]. В цьому напрямі було проведено аналогічні дослідження й багатьма іншими науковця, деякі з них спробуємо проаналізувати дещо детальніше.

У роботі [9] автор розглядає наявні стандарти та моделі управління якістю ПЗ. Також наводить особливості вдосконалення програмного процесу (англ. *Software Process Improvement*, SPI) [57, 61] розроблення ПЗ на підставі стандартів управління якістю та їх методології [5, 75]. Управління якістю ПЗ [9, 55] зосереджене на трьох основних стандартах сертифікації [31] та оцінювання [27, 29, 40, 41, 43, 46, 48, 71], а потім на двох додаткових методологіях управління якістю ПЗ: TickIt і Bootstrap. Автор роботи [9] вважає, що концепція SPI [5, 61] спрямована на вдосконалення процесу протягом життєвого циклу ПЗ [57, 75], добре відповідає принципу забезпечення його якості шляхом постійного вдосконалення процесу його розроблення. Стандарт ISO/IEC 90003 [44, 20] висвітлює особливості впровадження загальної методології управління якістю стандартів ISO 9001:2008 [31] для деяких випадків розроблення ПЗ та його підтримки. Стандарти ISO 9001:2015 стосуються розроблення, виготовлення, обслуговування та супроводу продукту проекту. Настанови з сертифікації продуктів проекту за стандартом ISO/IEC 90003 [20] підтверджують, що організація процесу розроблення та обслуговування ПЗ повністю відповідають його вимогам. Інститут розроблення ПЗ Університету Карнегі-Меллона (англ. *Carnegie Mellon University*, CMU) розробив моделі зрілості можливостей [76], коли опублікував перший короткий опис структури зрілості процесу розроблення ПЗ.

Згідно з визначенням, TickIT – це програма сертифікації для компаній, які займаються розробленням ПЗ, та для комп'ютерної галузі, яку підтримує переважно промисловість Великобританії та Швеції через UKAS та SWEDAC відповідно. Його загальна мета – покращити якість ПЗ. Водночас, Bootstrap – це клієнтський фреймворк, тобто інтерфейс для користувача, на відміну від коду серверної сторони, який знаходиться на сервері. Репозиторій із цим фреймворком є одним із найпопулярніших на GitHub. Серед інших, його використовують NASA і MSNBC.

У роботі [77] автори розглядають особливості гнучкого розроблення ПЗ, звертають увагу на сучасні тенденції, виклики та застосування цієї технології. Вони вважають, що така унікальна назва представляє весь спектр гнучких процесів розроблення ПЗ від фундаментальних концепцій до найвищих програмних рівнів, таких як аналіз вимог, тестування ПЗ, забезпечення його якості та управління ризиками [58, 73, 2, 78]. Гнучке розроблення ПЗ (англ. *Agile Software Development*, ASD) [3, 11, 54, 59, 77] стало популярною технологією, оскільки її методи можна застосувати до будь-якої парадигми програмування. Це важливо як для самого процесу розроблення ПЗ, оскільки такий підхід вимагає поетапної доставки замовнику продуктів проекту, так і на їхній командній співпраці під час його тестування, безперервному плануванні та навчанні користувачів порівняно з доставкою ПЗ наприкінці реалізації проекту всього й одразу.

Автори роботи [77] стверджують, що методологія Agile набула популярності завдяки використанню різних фреймворків, методів і технік, спроможних на покращення якості ПЗ. Водночас, методологія Scrum [6, 2] – основна гнучка структура, яку широко використовує спільнота розробників ПЗ. Вони вважають, що такі метаевристичні методи були використані в процесі гнучкого розроблення ПЗ для підвищення його якості, насамперед надійності. Ці методи не тільки покращують якість і надійність ПЗ, але й тестують різні сценарії його використання, що призводить до економічно ефективної реалізації проекту. Однак, щоб застосувати такі можливості ASD на практиці, необхідно вирішити багато важливих дослідницьких завдань [66]. З використанням різноманітних методів, керівних принципів, штучного інтелекту, програмного розрахунку та машинного навчання [77] такі завдання спрямовані насамперед на вивчення теоретичних і технологічних результатів проведених досліджень щодо всіх особливостей ASD. Окрім цього, відповідні дослідження проливають світло на останні тенденції, виклики та застосування ASD в різних областях ІТ-галузі.

У роботі [54] автори розглядають фактори та методи забезпечення якості ПЗ шляхом гнучкого його розроблення. Вони стверджують, що сучасні технології розроблення ПЗ розвиваються швидше, ніж будь-коли раніше, вимагаючи від розвинутих ІТ-компаній працювати в середовищі швидких змін. Автор роботи [54] вважає, що в сучасній бурхливій ІТ-індустрії вимоги до ПЗ можуть змінюватися щодня [30]. Щоб задовольнити ці поступово зростаючі вимоги, багато ІТ-компаній шукають нові методології розроблення ПЗ. Гнучкі методи його розроблення істотно змінили технологічний процес реалізації програмного проекту [45, 46, 47, 48]. Загальний термін Agile означає здатність швидко та терміново вносити зміни. Забезпечення якості ПЗ належить до систематичного процесу [52], який забезпечує досконалість програмних продуктів і наданих послуг. Автор роботи [54] стверджує, що його варто залучати до всього процесу Agile згідно з новою парадигмою – здатністю швидко та терміново вносити зміни [3, 11, 52, 54, 55]. У методології Agile кожен етап реалізації проекту аналізують, щоб мінімізувати кількість дій у будь-який момент часу. Забезпечення якості ПЗ під час його гнучкого розроблення створює передумови чіткого розуміння ключових концепцій, тенденцій і сучасних технологій реалізації проекту.

Отже, за результатами виконаної роботи можна сформулювати такі наукову новизну та практичну значущість результатів дослідження.

Наукова новизна отриманих результатів дослідження – розроблено ефективні SQA-процеси забезпечення якості продукту проекту, які потрібно виконувати та підтверджувати їх виконання, вимірювати та відстежувати відповідні дії, розробляти заходи для управління ними та їх вдосконалювати, а також заохочувати команду проекту використовувати ці процеси під час реалізації програмних проектів.

Практична значущість результатів дослідження – результати виконання SQA-процесів відповідно до настанов стандарту IEEE 730-2014 веб-додаток було представлено безпосередньому користувачу для того, щоб зробити його доступним для виявлення дефектів під час реального його використання у виробничих умовах.

Висновки / Conclusions

Проаналізовано наявні підходи до вирішення проблеми забезпечення якості програмного продукту в межах життєвого циклу реалізації проекту, розглянуто та проаналізовано ефективність застосування цих підходів під час тестування реального веб-додатку "Інтернет-багкінг", де звернуто увагу насамперед на етапи розроблення відповідних тестів, процеси забезпечення його якості та особливості розгортання. За результатами виконаного дослідження можна зробити такі основні висновки.

1. Проаналізовано настанови стандарту IEEE 730-2014, які дають можливість гарантувати якість програмного продукту, а також настанови стандарту ISO/IEC 12207:2018, які забезпечують якість ПЗ протягом його життєвого циклу та особливості застосування цих настанов на практиці. Встановлено, що гарантія якості програмного продукту є обов'язковою для будь-якого IT-бізнесу, незалежно від того, чи це продукт для виконання повсякденних операцій, чи для споживачів критичної інфраструктури. Завдання перевірки якості, викладені в стандарті IEEE 12207 для процесів життєвого циклу ПЗ, гарантують якість його розроблення та ефективність використання користувачами.

2. З'ясовано, що стандарт IEEE 730-2014 описує SQA-процеси, тобто комплексну діяльність, яку використовують протягом усього життєвого циклу ПЗ. Позаяк існують різні види SQA-діяльності, то стандарти та відповідні методи гарантують ПЗ належну його якість, що відповідатиме потребам замовника. Визначено ефективні SQA-процеси забезпечення якості продукту проекту, які потрібно не тільки виконувати, а й підтверджувати їх виконання, як вимірювати та відстежувати відповідні дії, як розробляти заходи для управління ними та їх вдосконалювати, а також як заохочувати команду проекту використовувати ці процеси під час реалізації програмних проектів.

3. Встановлено, що для досягнення цілей реалізації проекту потрібно запроваджувати координацію та верифікацію продукту проекту, його підтвердження та перегляд, аудит та інші настанови стандарту ISO/IEC 12207:2018. Виходячи з цих вказівок, було сформовано структуру SQA-процесів для веб-додатку "Інтернет-Банкінг", який було протестовано в ході виконання дослідження. З'ясовано, що кожен програмний проект має свої особливості запровадження процесів забезпечення якості продукту проекту. В банківському додатку надзвичайно важливо проводити тестування ПЗ на кібербезпеку, цілісність клієнтських даних, спостережність транзакцій, що і було описано в даній роботі.

4. Після виконання SQA-процесів відповідно до настанов стандарту IEEE 730-2014 продукт проекту "Інтернет-банкінг" було представлено безпосередньому користувачу. Розгортання ПЗ містило дії, необхідні для того, щоб зробити його доступним для виявлення дефектів під час реального його використання у виробничих умовах. Виявлено, що ефективна система контролю якості ПЗ забезпечує кращий досвід роботи з безпосередніми користувачами і збільшує тривалість його безвідмовної роботи. Навіть поодинокі дефекти, виявлені в роботі ПЗ, потенційно впливають на ключові бізнес-процеси замовника, що призведе до значних втрат прибутку. Запобігання цьому – шанс створити імідж бренду компанії-розробнику ПЗ, якій довіряють.

References

1. Batenko, L. P., Zagorodnih, O. A., & Lishchinska, V. V. (2003). Management of software projects: a study guide. Kyiv: KNEU, 231 p. [In Ukrainian].
2. Breno, G. Tavares, Carlos, Eduardo S. da Silva, & Adler, D. de Souza. (2017). Risk Management in Scrum Projects: A Bibliometric Study. *Journal of communications software and systems*, No. 13(1), 25–41. https://doi.org/10.24138/jcomss.v13i1.241
3. Chugh, M., & Chugh, N. (2023). A Deep Drive into Software Development Agile Methodologies for Software Quality Assurance. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal & M. Sood). https://doi.org/10.1002/9781119896838.ch12
4. Clarke, P., & O'Connor, R. V. (2010). Harnessing ISO/IEC 12207 to Examine the Extent of SPI Activity in an Organisation. In: Riel, A., O'Connor, R., Tichkiewitch, S., Messnarz, R. (eds) Systems, Software and Services Process Improvement. EuroSPI 2010. Communications in Computer and Information Science, Vol. 99. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-15666-3_3
5. Clarke, P., O'Connor, R. V., & Yilmaz, M. (2012). A Hierarchy of SPI Activities for Software SMEs: Results from ISO/IEC 12207-Based SPI Assessments. Proceedings of the 12th International Conference on Software Process and Capability Determination. (SPICE'12), CCIS Vol. 290, Springer, Berlin, Germany, 62–74. https://doi.org/10.1007/978-3-642-30439-2_6
6. Diaz, J., Garbajosa, J., & Calco-Manzano, J. A. (2009). Mapping CMMI Level 2 to Scrum Practices: An Experience Report. In: O'Connor, R. V., Baddoo, N., Cuadrago Gallego, J., Rejas Muslera, R., Smolander, K., Messnarz, R. (Eds). Software Process Improvement. EuroSPI 2009. Communications in Computer and Information Science, Vol. 42, No. 2, 93–104. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-04133-4_8
7. Egler, Miguel. (2019). Testing for the phenomenal: Intuition, Metacognition, and philosophical Methodology. Mind & Language, Vol. 35, Issue 1, 48–66. https://doi.org/10.1111/mila.12229
8. Galin, D. (Ed.). (2018). Software Development and Quality Assurance Process Standards. In Software Quality: Concepts and Practice. https://doi.org/10.1002/9781119134527.app1
9. Galin, D. (Ed.). (2018). Software Quality Management Standards and Models. In Software Quality: Concepts and Practice. https://doi.org/10.1002/9781119134527.app2
10. GMP Navigator. (2023). FDA Guidance for Industry, FDA Reviewers and Compliance on Off-The-Shelf Software Use in Medical Devices. URL: https://www.gmp-navigator.com/files/guidemgr/585.pdf
11. Greer, D., & Hamon, Y. (2011). Agile Software Development. Software: Practice and Experience, 41, 943–944. https://doi.org/10.1002/spe.1100
12. Guey-Shin, C., Horng-Linn, P., & Jer-Nan, J. (2008). A review of systems engineering standards and processes. *Journal of Biomechanics Engineering*, Vol. 1, No. 1, 71–85. URL: http://journal.tibm.org.tw/wp-content/uploads/2013/06/4.-systems-engineering-standards-and-processes.pdf
13. Guidance Document. (2005). Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices. Guidance for Industry and FDA Staff. URL: https://www.fda.gov/regulatory-information/search-fda-guidance-documents/guidance-content-premarket-submissions-software-contained-medical-devices
14. IEC. (1996). IEC 60601-1-4:1996. Medical electrical equipment – Part 1–4: General requirements for safety – Collateral Standard:

programmable electrical medical systems. URL: https://webstore.iec.ch/publication/16790
15. IEC. (2010). IEC 61508:2010. Functional safety of electrical/electronic/programmable electronic safety-related systems – Part 1-4: General requirements (see Functional Safety and IEC 61508). URL: https://webstore.iec.ch/publication/5515
16. IEC. (2011). IEC 61513:2011. Nuclear power plants – Instrumentation and control important to safety – General requirements for systems. URL: https://webstore.iec.ch/publication/5532
17. IEEE SA. (2005). IEEE 1220-2005. IEEE Standard for Application and Management of the Systems Engineering Process. URL: https://standards.ieee.org/ieee/1220/3372/
18. IEEE SA. (2008). IEEE 15939-2008. IEEE Standard Adoption of ISO/IEC 15939:2007 – Systems and Software Engineering – Measurement Process. URL: https://standards.ieee.org/ieee/15939/4262/
19. IEEE SA. (2008). IEEE 829-2008. IEEE Standard for Software and System Test Documentation. URL: https://standards.ieee.org/ieee/829/3787/
20. IEEE SA. (2008). IEEE 90003-2008. IEEE Guide – Adoption of ISO/IEC 90003:2004 Software Engineering – Guidelines for the Application of ISO 9001:2000 to Computer Software. URL: https://standards.ieee.org/ieee/90003/3677/
21. IEEE SA. (2010). IEEE 7-4.3.2-2010. IEEE Standard Criteria for Digital Computers in Safety Systems of Nuclear Power Generating Stations. URL: https://standards.ieee.org/ieee/7-4.3.2/4834/
22. IEEE SA. (2012). IEEE 1012–2012. IEEE Standard for System and Software Verification and Validation. The IEEE Computer Society, IEEE, New York. URL: https://standards.ieee.org/ieee/1012/4021
23. IEEE SA. (2014). IEEE 730-2014–2014. IEEE Standard for Software Quality Assurance Processes. The IEEE Computer Society, IEEE, New York. URL: https://standards.ieee.org/ieee/730/5284/
24. IEEE Standard for Software Quality Assurance Processes. (2014). In IEEE 730-2014-2014 (Revision of IEEE 730-2014-2002), Vol., No., pp. 1–138, 13 June 2014, https://doi.org/10.1109/IEEESTD.2014.6835311
25. ISO. (2003). ISO 13485:2003. Medical devices – Quality management systems – Requirements for regulatory purposes. URL: https://www.iso.org/standard/36786.html
26. ISO. (2007). ISO 14971:2007. Medical devices – Application of risk management to medical devices. URL: https://www.iso.org/standard/38193.html
27. ISO/IEC. (2003). ISO/IEC 15504-2:2003. Software Engineering – Process Assessment – Part 2 – Performing an Assessment. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/37458.html
28. ISO/IEC. (2004). ISO/IEC 15504-1:2004. Information Technology – Process Assessment – Part 1: Concepts and Vocabulary. URL: https://www.iso.org/ru/standard/38932.html
29. ISO/IEC. (2004). ISO/IEC 15504-3:2004. Information Technology – Process Assessment – Part 3 – Guidance on Performing an Assessment. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/37454.html
30. ISO/IEC. (2004). ISO/IEC 15504-4:2004. Information Technology – Process Assessment – Part 4 – Guidance on Use for Process Improvement and Process Capability Determination. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/37462.html
31. ISO/IEC. (2008). ISO 9001:2008. Quality management systems – Requirements. This standard has been revised by ISO 9001:2015, Geneva. URL: https://www.iso.org/standard/46486.html
32. ISO/IEC. (2008). ISO/IEC 12207:2008. Systems and Software Engineering – Software Life Cycle Processes, ISO – International Organization for Standardization, Geneva, Switzerland. URL: https://www.iso.org/standard/43447.html
33. ISO/IEC. (2010). ISO/IEC TR 24774:2010. Systems and Software Engineering – Life Cycle Management – Guidelines for Process Description. URL: https://www.iso.org/standard/53815.html
34. ISO/IEC. (2011). ISO/IEC 15026-2:2011. Systems and software engineering – Systems and software assurance – Part 2: Assurance case. URL: https://www.iso.org/standard/52926.html
35. ISO/IEC. (2011). ISO/IEC 15026-3:2011. Systems and software engineering – Systems and software assurance – Part 3: System integrity levels. URL: https://www.iso.org/standard/57107.html
36. ISO/IEC. (2011). ISO/IEC 29110-1:2011. Software engineering — Lifecycle profiles for Very Small Entities (VSEs) – Part 1: Overview. URL: https://www.iso.org/ru/standard/51150.html
37. ISO/IEC. (2011). ISO/IEC TS 15504-10:2011. Information Technology – Process Assessment – Part 10 – Safety Extension. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/54537.html
38. ISO/IEC. (2011). ISO/IEC TS 15504-9:2011. Information Technology – Process Assessment – Part 9 – Target process Profiles. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/51684.html
39. ISO/IEC. (2012). ISO/IEC 15026-4:2012. Systems and software engineering – Systems and software assurance – Part 4: Assurance in the life cycle. URL: https://www.iso.org/standard/59927.html
40. ISO/IEC. (2012). ISO/IEC 15504-5:2012. Information Technology – Process Assessment – Part 5 – An Exemplar Software Life Cycle Process Assessment Model. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/60555.html
41. ISO/IEC. (2012). ISO/IEC 15504-8:2012. Information Technology – Process Assessment – Part 8 – An Exemplar Process Assessment Model for IT Service Management. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/50625.html
42. ISO/IEC. (2013). ISO/IEC 15026-1:2013. Systems and Software Engineering – Systems and software Assurance – Part 1: Concepts and Vocabulary. URL: https://www.iso.org/standard/62526.html
43. ISO/IEC. (2013). ISO/IEC 15504-6:2013. Information Technology – Process Assessment – Part 6 – An Exemplar System Life Cycle Process Assessment Model. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/61492.html
44. ISO/IEC. (2014). ISO/IEC 90003:2014. Software Engineering – Guidelines for the Application of ISO 9001:2008 to Computer Software. International Organization for Standardization (ISO), Geneva, Switzerland. URL: https://www.iso.org/standard/66240.html
45. ISO/IEC. (2015). ISO/IEC 33001:2015. Information Technology – Process Assessment – Concepts and Terminology. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/54175.html
46. ISO/IEC. (2015). ISO/IEC 33002:2015. Information Technology – Process Assessment – Requirement for Performing Process Assessment. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/54176.html
47. ISO/IEC. (2015). ISO/IEC 33003:2015. Information Technology – Process Assessment – Requirement for Process Measurement Frameworks. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/54177.html
48. ISO/IEC. (2015). ISO/IEC 33004:2015. Information Technology – Process Assessment – Requirements for Process Reference, Process Assessment and Maturity Models. International Organization for Standardization (ISO), Geneva. URL: https://www.iso.org/standard/54178.html
49. ISO/IEC. (2015). ISO/IEC 33014:2015, IDT. Information Technology – Process Assessment – Guide for Process Improvement. International Organization for Standardization (ISO), Geneva. URL: http://online.budstandart.com/ua/catalog/doc-page.html?id_doc=69111
50. ISO/IEC. (2018). ISO/IEC 12207:2018. Systems And Software Engineering – Software Life Cycle Processes (ISO/IEC/IEEE 12207:2017, IDT) (Swedish Standard). URL: https://webstore.ansi.org/standards/sis/ssisoiecieee122072018

51. ISO/IEC. (2019). ISO/IEC/IEEE 15289:2019. Systems and software engineering — Content of life-cycle information items (documentation). URL: https://www.iso.org/standard/74909.html
52. Jadhav, D., Kundale, J., Bhagwat, S., & Joshi, J. (2023). A Systematic Review of the Tools and Techniques in Distributed Agile Software Development. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal & M. Sood). https://doi.org/10.1002/9781119896838.ch8
53. Katrenko, A. V. (2011). IT project management. Lviv: Novy svit-2000, 540 p. [In Ukrainian].
54. Kaur, G., Kaur, I., Harnal, S., & Malik, S. (2023). Factors and Techniques for Software Quality Assurance in Agile Software Development. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal & M. Sood). https://doi.org/10.1002/9781119896838.ch13
55. Kaur, S., Hooda, S., & Deo, H. (2023). Software Quality Management by Agile Testing. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal & M. Sood). https://doi.org/10.1002/9781119896838.ch11
56. Khanna, E., Popli, R., & Chauhan, N. (2023). Classification of Risk Factors in Distributed Agile Software Development Based on User Story. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal and M. Sood). https://doi.org/10.1002/9781119896838.ch14
57. Krishnamurthy A., & O'Connor R. V. (2013). Using ISO/IEC 12207 to Analyze Open Source Software Development Processes: An E-Learning Case Study. In: Woronowicz, T., Rout, T., O'Connor, R.V., Dorling, A. (Eds). Software Process Improvement and Capability Determination. (SPICE 2013). Communications in Computer and Information Science, Vol. 349, Springer, Berlin, Heidelberg, 1–12. https://doi.org/10.1007/978-3-642-38833-0_10
58. Krishnan, Soumya M. (2015). Software Development Risk Aspects and Success Frequency on Spiral and Agile Model. International *Journal of Innovative Research in Computer and Communication Engineering*, 3(1), 301–310. https://doi.org/10.15680/ijircce.2015.0301024
59. Kruchten, P. (2013). Contextualizing Agile Software Development. Journal of Software: Evolution and Process, 25, 351–361. https://doi.org/10.1002/smr.572
60. Kuzmin, O. E., Podolchak, N. Yu., & Matviishyn, V. E. (2011). Management and reduction of energy supply risks of enterprises: monograph; National Lviv Polytechnic University. Lviv: City Information Systems, 235 p. [In Ukrainian].
61. Laporte C. Y., Alexandre S., & O'Connor R. V. (2008). A software engineering lifecycle standard for very small enterprises, Software Process Improvement. Communications in Computer and Information Science, Vol. 16, No. 2, 129–141.
62. Laporte C. Y., April A., & Renault A. (2006). Applying ISO/IEC software engineering standards in small settings: historical perspectives and initial achievements. In Proceedings of SPICE 2006 Conference, Luxembourg, May 2006, 1–5.
63. Lukyanova, V. V. (2007). Risk diagnostics of IT company activity: monograph. Khmelnytskyi: Private entrepreneur V. V. Kovalskyi, 312 p. [In Ukrainian].
64. Mishra, D., & Mishra, A. (2011), Complex software project development: agile methods adoption. *Journal of Software Maintenance and Evolution: Research and Practice*, 23: 549–564. https://doi.org/10.1002/smr.528
65. Moore, J. W. (2002). Software Engineering Standards. In Encyclopedia of Software Engineering, J. J. Marciniak (Ed.). https://doi.org/10.1002/0471028959.sof319
66. Morales Trujillo, M., Oktaba, H., Pino, F. J., & Orozco, M. J. (2011). Applying Agile and Lean Practices in a Software Development Project into a CMMI Organization. In: Caivano, D., Oivo, M., Baldassarre, M.T., Visaggio, G. (Eds). Product-Focused Software Process Improvement. PROFES 2011. Lecture Notes in Computer Science, Vol. 6759. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-21843-9_4
67. Myers, G. J., Badgett, T., & Sandler, C. (Eds). (2012). Testing in the Agile Environment. In The Art of Software Testing. Part 9. https://doi.org/10.1002/9781119202486.ch9
68. Nozdrina, L. V., Yashchuk, V. I., & Polotai, O. I. (2010). Management of software projects: a textbook for university students. Kyiv: Center for Educational Literature, 430 p. [In Ukrainian].
69. Rafiq Ahmad Khan, & Siffat Ullah Khan. (2018). A preliminary structure of software security assurance model. *Proceedings of the 13th Conference on Global Software Engineering*, ICGSE '18, 137–140. https://doi.org/10.1145/3196369.3196385
70. Rashid, N., & Khan, S. U. (2018). Using agile methods for the development of green and sustainable software: Success factors for GSD vendors. *Journal of Software: Evolution and Process*. https://doi.org/10.1002/smr.1927
71. Salviano, C. F., Alves, A., Stefanuto, G. N., Maintinguer, S. T., Mattos, C. V., Zeitoum, C., & Reuss, G. (2012). Developing a Process Assessment Model for Technological and Business Competencies on Software Development. *Eighth International Conference on the Quality of Information and Communications Technology*, Lisbon, Portugal, 125–130. https://doi.org/10.1109/QUATIC.2012.27
72. Samli, Gupta, M., Sharma, A., Hooda, S., & Bhatia, J. S. (2023). Distributed Agile Software Development (DASD) Process. In Agile Software Development (Eds S. Hooda, V. M. Sood, Y. Singh, S. Dalal & M. Sood). https://doi.org/10.1002/9781119896838.ch9
73. Sandra, M. N., Sandra, M. N., & Carlos, S. S. (2016). Risk management applied to software development projects in incubated technology-based companies: literature review, classification, and analysis. Gest. Prod., Sao Carlos, Vol. 23(4), 798–814. https://doi.org/10.1590/0104-530X472-15
74. SDLC. (2023). 6 main stages of the SDLC. Online IT Education. QA. QC. Software Testing. Knowledge. Practice. Job. Career. URL: https://qagroup.com.ua/publications/6-osnovnykh-etapiv-sdlc/
75. Stallinger, F., & Neumann, R. (2012). Extending ISO/IEC 12207 with Software Product Management: A Process Reference Model Proposal. In: Mas, A., Mesquida, A., Rout, T., O'Connor, R.V., Dorling, A. (Eds). Software Process Improvement and Capability Determination. SPICE 2012. Communications in Computer and Information Science, Vol. 290, Springer, Berlin, Germany, 93–106. https://doi.org/10.1007/978-3-642-30439-2_9
76. Suominen, M., & Makinen, T. (2014). On the applicability of capability models for small software organizations: does the use of standard processes lead to a better achievement of business goals? Software Quality Journal, Vol. 22, No. 4, 579–591. https://doi.org/10.1007/s11219-013-9201-7
77. Susheela Hooda, Vandana Mohindru Sood, Yashwant Singh, Sandeep Dalal, & Manu Sood. (2023). Agile Software Development: Trends, Challenges and Applications. Scrivener Publishing LLC. https://doi.org/10.1002/9781119896838
78. Yet, B., Constantinou, A., Fenton, N., Neil, M., Luedeling, E., & Shepherd, K. (2016). A Bayesian network framework for project cost, benefit and risk analysis with an agricultural development case study. Expert Systems with Applications, Vol. 60, Pages 141–155. https://doi.org/10.1016/j.eswa.2016.05.005

P. Yu. Grytsiuk, A. V. Ivanyshyn, Yu. I. Hrytsiuk
Lviv Polytechnic National University, Lviv, Ukraine

QUALITY ASSURANCE OF SOFTWARE PRODUCTS IN ACCORDANCE WITH IEEE 730-2014 STANDARD WITHIN THE PROJECT IMPLEMENTATION LIFECYCLE

The available approaches for addressing the issue of software product quality assurance within the project implementation lifecycle were analyzed. The effectiveness of applying relevant standards for testing a real web application was considered and

analyzed, with attention given to the stages of developing corresponding tests, quality assurance processes, and deployment peculiarities. The guidelines of IEEE 730-2014 standard were considered, which enable guaranteeing the quality of software products, as well as ISO/IEC 12207:2018 standard, which ensures the quality of software throughout its lifecycle, and their practical application features were analyzed. It was established that software product quality assurance is mandatory for any IT business, regardless of whether it is a product for everyday operations or for critical infrastructure consumers. The quality verification tasks outlined in the IEEE 12207:2018 standard for software lifecycle processes ensure the quality of its development and effectiveness of its use by users. The standard IEEE 730-2014 is found to describe SQA processes that must be used throughout the entire software life cycle. The existence of various types of SQA activities, laid down in the relevant standards, ensures that the software quality will meet the customer's needs. Effective SQA processes to ensure project product quality have been identified, which not only need to be executed but also confirmed, with features for measuring and tracking the relevant processes, rules for developing measures to manage and improve them, and approaches to encourage the project team to use SQA processes during software projects implementation. It has been established that to achieve the goals of project implementation, coordination, product verification, confirmation, review, audit, and other guidelines from the ISO/IEC 12207:2018 standard must be implemented. Based on these guidelines, a structure of SQA processes was formed for the web application "Online-Banking," which was tested during the study. It was found that each unique software project has its own peculiarities when it comes to implementing product quality assurance processes. In a banking application, it is extremely important to conduct software testing for cyber security, integrity of client data, transaction observability, which were described in this study. After completing all SQA processes, the "Online-Banking" project product was presented to the end user. Software deployment included actions necessary to make it available to the relevant users. It was found that an effective software quality control system ensures a better user experience and increases its faultless operation time. At the same time, even single defects found in the software's operation can affect the customer's key business processes, resulting in significant revenue losses. Preventing this is an opportunity to create a brand image for the software development company that customers trust.

Keywords: quality standards of software products; software quality assurance; testing; SQA activities; requirements specification; customer requirements; quality control.