УДК 004.415.532

О. А. РЕМІННИЙ

Вінницький національний технічний університет, м. Вінниця

ПАТЕРНИ АВТОМАТИЗОВАНОГО ФУНКЦІОНАЛЬНОГО ТЕСТУВАННЯ КОРИСТУВАЦЬКИХ ІНТЕРФЕЙСІВ

Анотація. Проведено класифікацію дизайн патернів функціонального автоматизованого тестування. Вперше виділено групи патернів функціонального автоматизованого тестування, до яких приводить багатошарова архітектура системи тестування. Зроблена класифікація патернів за їх призначенням. Виділено архітектурний архетип системи автоматизованого тестування у вигляді багатошарового композитного рішення з високою ізольованістю шарів та мета фреймворком.

Ключові слова: класифікація патернів, користувацький інтерфейс, тестування.

Аннотация. Проведена классификация дизайн паттернов функционального автоматизированного тестирования. Впервые выделены группы паттернов функционального автоматизированного тестирования, к которым приводит многослойная архитектура системы тестирования. Произведена классификация паттернов по их назначению. Выделено архитектурный архетип системы автоматизированного тестирования в виде многослойного композитного решения с высокой изолированностью слоев и мета фреймворком.

Ключевые слова: классификация паттернов, пользовательский интерфейс, тестирование.

Abstract. Made classification of functional automated test design patterns. Patterns are grouped into categories caused by a multilayer architecture of the test system. Made classification of patterns according to their purpose. Allocated architectural archetype of automated testing as a multilayered composite solution with high layers isolation and reusable meta framework block.

Keyworld: classification of patterns, user interface, testing.

Вступ

Зараз існує досить широка кількість підходів до розробки програмного забезпечення: Об’єктно орієнтоване програмування, функціональне програмування, багато їх піднапрямків (Domain Driven Design, Test Driven Design, Behavior Driven Design та ін.) Дані підходи мають в основі певні декларативні концепції та теореми, які спрощують процеси визначення початкової архітектури системи, розуміння системи, передачу знань між розробниками і т.п.

Процес розробки рішення комплексу автоматизованих функціональних тестів певної програми не сильно відрізняється від процесу створення самої програми. Автоматизоване тестування є досить молодим напрямком і на даний момент швидко розвивається, стандартизується та удосконалюється. Створюються нові утиліти для взаємодії з тестованою системою (SUT – System Under Test).

На даний момент досить критичною частиною розробки автоматизованих тестів став етап підготовки та визначення базових практик та підходів, за допомогою яких буде розроблятись система автоматизованого тестування. На рис 1 можна побачити як взаємодіють між собою система автоматизованого тестування та тестер:

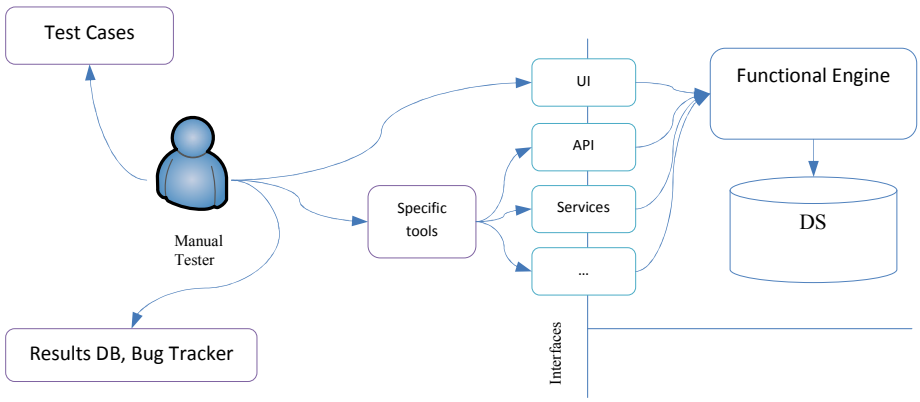

Рисунок 1 - Взаємодія тестера з тестованою системою

Центральною частиною такої системи є особа тестер (Manual Tester). Використовуючи ручну взаємодію та візуальний аналіз аплікації, а також спеціальні утиліти доступу (Specific tools) до невізуальних інтерфейсів тестованої системи, тестер відтворює викладені в тест кейсах (Test Cases) сценарії. У випадку збою чи непрогнозованої поведінки системи, тестер додає інформацію про некоректну поведінку в систему відстеження помилок системи (Bug tracker).

Основною ціллю автоматизованого тестування є видалення людської взаємодії з тестованою системою. З огляду літературних джерел видно, що на ринку присутня досить велика кількість систем автоматизованого тестування. Як виявив літературний пошук [1], на даний момент існує більше 20 комерційних та безкоштовних продуктів які дозволяють доступатись до інтерфейсів системи (interfaces), і які декларують певні конкретні вимоги та рекомендації по роботі конкретно з їхнім продуктом. Однак немає чітко відокремлених практик, які були б незалежними від постачальника продукту автоматизованого тестування.

Також постачальники програмних продуктів досить часто вдаються до маркетингових хитрощів і описують переваги їхніх систем при малій кількості функціональних тестів, а як показує практика найдорожчою в процесі роботи з системою виявляється підтримка існуючих тестів. А для того щоб правильно розробити фреймворк автоматизованого тестування, потрібно керуватись ні від чого незалежним набором правил та практик.

Мета

Основною задачею даної статті є виділення набору базових патернів автоматизованого функціонального тестування графічних інтерфейсів програмних додатків та їх групування за призначенням.

Постановка задачі

Перед тим як перейти до визначення набору патернів, звернемось до самого поняття патернів. Термін патерн дизайну вперше був визначений архітектором Крістофером Олександром.

"Кожен патерн описує проблему, яка виникає знову і знову в нашому середовищі, а потім описує основні рішення цієї проблеми, таким чином, що ви можете використовувати це рішення мільйон разів, ніколи не вирішуючи ті ж самі задачі".

Одним з основоположників патернів дизайну в програмуванні можна вважати групу співавторів GOF [2]. Відповідно їх визначення:

"Кожен патерн дизайну систематично окреслює, пояснює і оцінює важливі повторювані дизайни в об'єктно-орієнтованій системі".

Далі перейдемо до визначення самих патернів по категоріях. Як приклад для застосування розглянемо наступну задачу автоматизації веб сайту (рис 2). Додаток має в собі набір сторінок, серед яких є логін сторінка. Кожен тест має пройти через логін сторінку для виконання подальшого функціоналу.

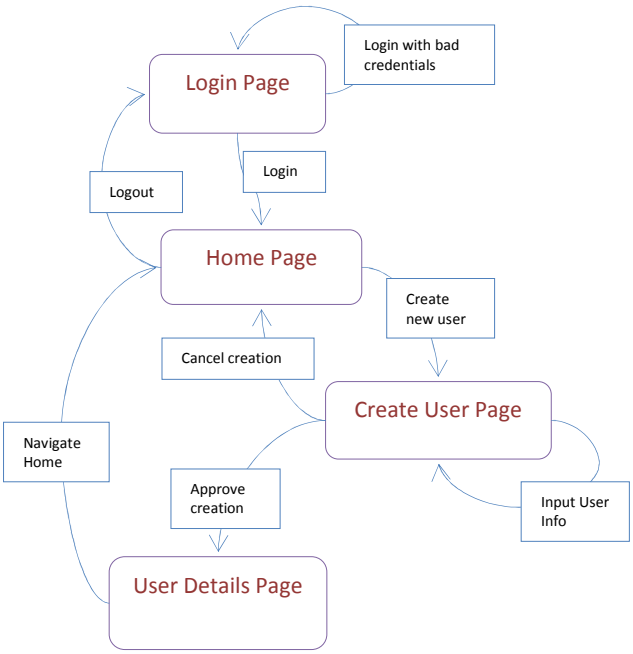

Рисунок 2 – Приклад простої веб-аплікації з мінімальним набором сторінок та функціоналу

Патерни автоматизованого функціонального тестування

На рисунку 3 наведена загальна класифікаційна схема патернів автоматизованого тестування. Вони розбиті на групи відповідно до тих цільових задач. Далі наведено більш детальний опис кожної категорії та самих шаблонів.

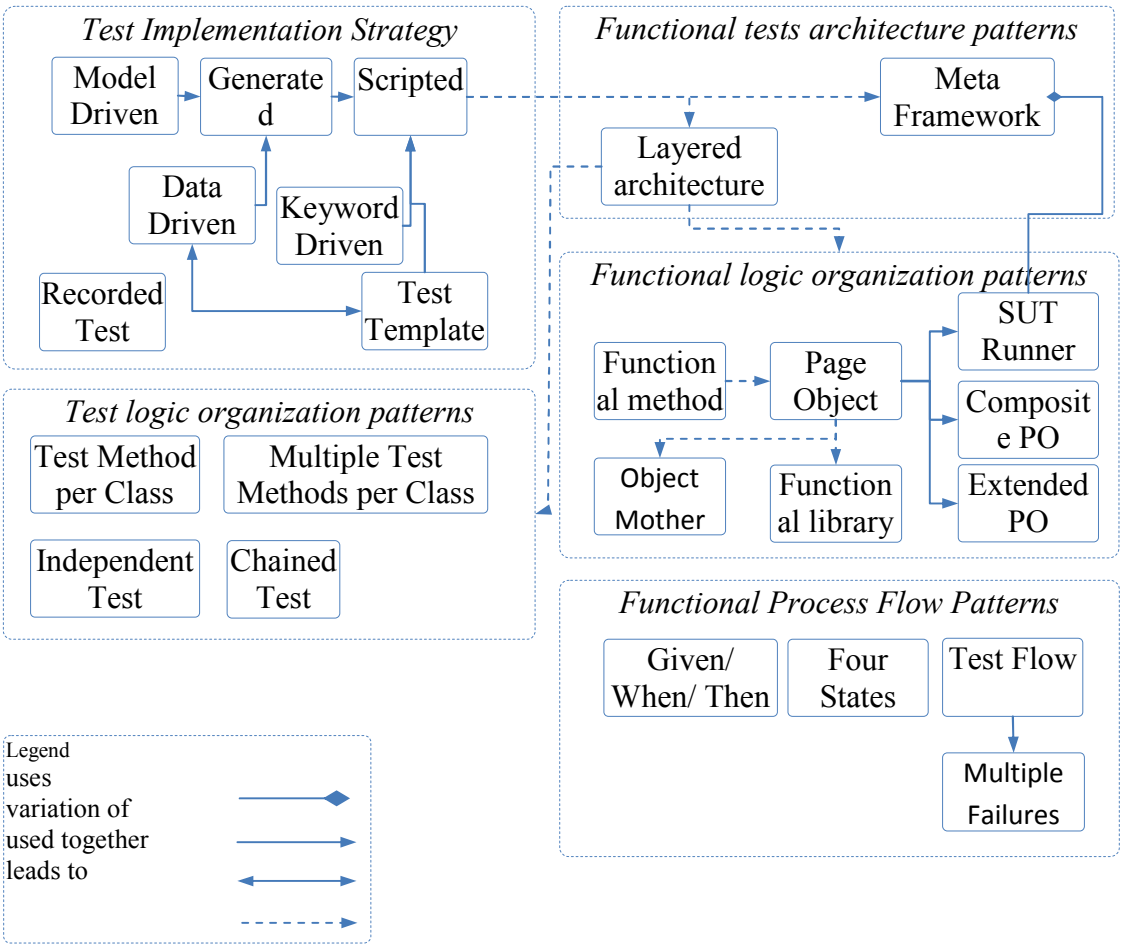

Рисунок 3 - Класифікаційна схема шаблонів автоматизованого функціонального тестування

Патерни імплементації тестів

Дана категорія об'єднує набір патернів які дозволяють обрати стратегію розробки рішення автоматизованого тестування.

Записані тести (Recorded). Імплементація тесту виконується за рахунок утиліти автоматизованого тестування, яка вміє виконувати запис та відтворення. В деякій мірі вважається поганою практикою, так як дуже дорогі в підтримці.

Запрограмовані (Scripted). Імплементація тесту виконується програмістом.

Імплементація базового шаблону (Test Template). Імплементація базового класу шаблону тесту. Варіації тесту – за рахунок наслідування та розширення функціоналу шаблону.

Імплементація за даними (Data Driven). Імплементація базового тесту визначається тест кейсом. Варіації тесту – за рахунок набору ріних вхідних комбінацій даних.

Імплементація за Ключовими словами (Keyword Driven). Імплементація тесту за допомогою ключових слів (як Натиснути, Ввести і т.п.).

На даний момент існує декілька програмних утиліт, які дозволяють імплементувати тести за допомогою ключових слів. Кроки тесту мають вигляд набору з ключового слова, назви контролу на екрані та вхідних параметрів.

Імплементація з моделі (Model Driven). Будь яка аплікація в конкретний момент часу з конкретними вхідними даними може знаходитись тільки в одному конкретному коректному стані. Відповідно до такого визначення можна стверджувати що програма є кінцевою машиною станів (кінцевим автоматом). Враховуючи цей факт та наявність моделі станів та переходів (прикладом є рисунок X) можна визначити певні набори переходів між сторінками які б максимально покривали функціонал програми.

Архітектурні патерни

Багатошарова структура тестової системи (Multi Layered Test Solution). Розділяє логіку тестової системи на окремі логічні шари.

Це добре розповсюджена ідея - розподілену систему архітектурно розділяти на окремі шари. Перший рівень інкапсулює логіку представлення, другий - рівень бізнес-логіки, а третій шар відповідає

за зберігання даних. Використання цієї парадигми дозволяє знизити вартість обслуговування програмних додатків, так як компоненти всередині кожного рівня можуть бути змінені без впливу на інші рівні. Той же принцип може бути застосований до системи тестування.

Код тестів можна розділити на три шари: шар інструменту доступу до користувацьких інтерфейсів програмного додатку, шар функціональної логіки та шар тест кейсу. Кожен шар має певну відповідальність із загальною метою скорочення витрат на підтримку тестів і полегшення створення нових тестів.

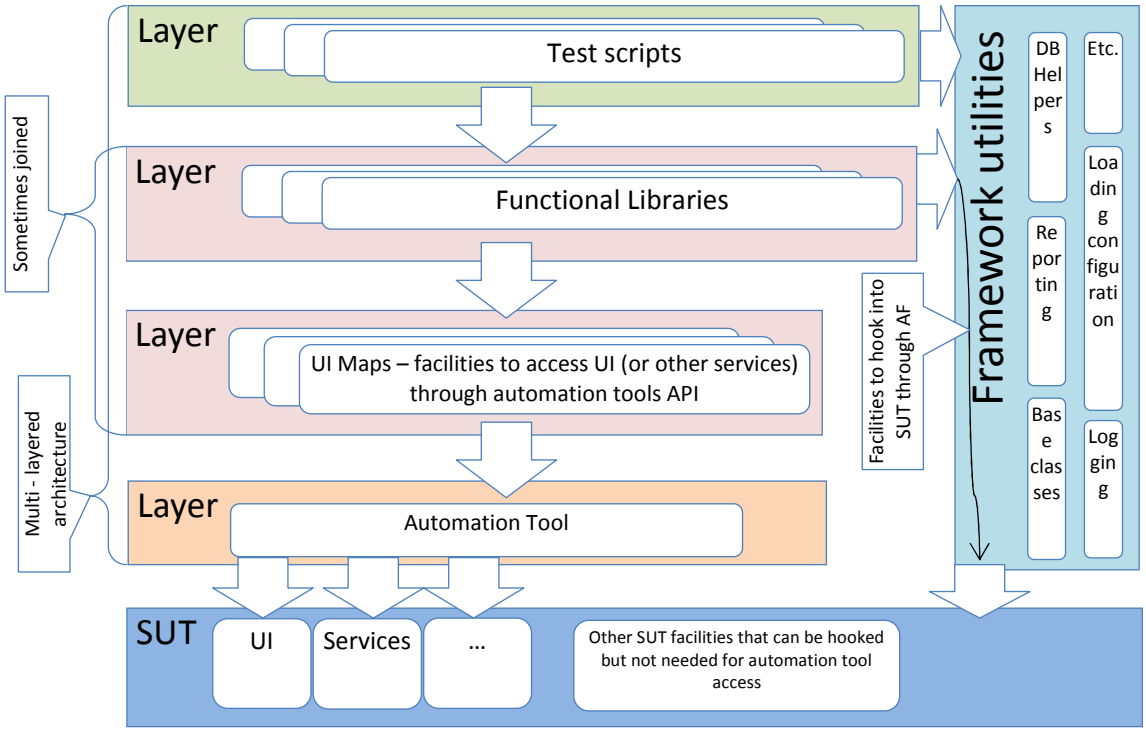

Рисунок 4 – Архітектурний архетип - багатошарова архітектура тестової системи

Мета Фреймфорк (Meta Framework). Виділяє набір базових незалежних утилітарних класів, які можуть використовуватись незалежно від обраних інструментів тестування та перевикористовуватись між проектами.

Такі рішення зазвичай потрібні у випадку коли тестуються різні проекти всередині однієї корпорації і корпоративний стандарт вимагає уніфікованого інтерфейсу результатів. Також Мета фреймворк покращує показники перевикористання коду між проектами, оскільки кращі утилітні методи можуть включатись до його складу. Базові класи як для функціональних так і тестових об’єктів спрощують передачу знань між різними проектами. На рисунку Х Мета фреймворк виділено в правій частині.

Функціональні композиційні патерни

Дана група шаблонів описує як можна виділяти частини коду в універсальні одиночні компоненти, призначені для багаторазового використання.

Функціональний Метод (Functional method). Абстрагує конкретну функцію аплікації від її реалізації на UI, API чи іншому рівні.

Багато утиліт для автоматизованого тестування дозволяють створювати так звані «записані сценарії» - коли розробник тесту робить певні дії з конкретною аплікацією і вони автоматично створюють скрипт тесту, який потім можна буде повторно програти і перевірити чи він виконався коректно після змін в програмі.

Зміна вигляду логін сторінки призведе до того, що потрібно буде провести зміни в усіх пропонованих методах. Якщо ж абстрагувати логін метод у вигляді Application.Login(username, password), і у всіх тестах використовувати саме цей метод, то у випадку зміни логін сторінки нам потрібно буде модифікувати лише один функціональний метод, і з зміни автоматично розповсюдяться на всі тести де цей метод використовується.

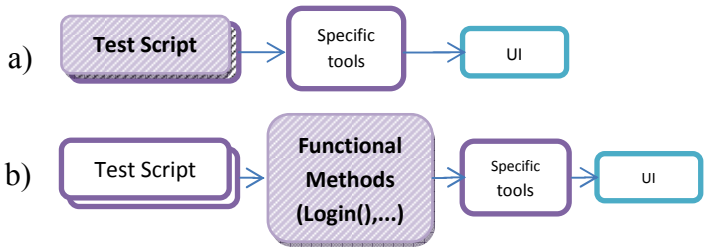

Рисунок 5 - Взаємодія тестового скріпту з користувацьким інтерфесом без проміжного шару функціональних методів (а) та з шаром функціональних методів (б)

Об'єкт сторінки (Page Object). Групує функціональні методи певної сторінки.

Функціональні методи для описаної на рисунку X аплікації можна винести в єдиний клас, так як їх є небагато. Однак для покращення підтримуваності коду буде краще згрупувати методи відповідно до сторінок, які представляють ці методи: сторінка PageLogin, методи: Login(); сторінка PageHome: методи: Logout(), CreateUser().

Функціональна бібліотека (Functional library) Групує функціональні об'єкти або (і) функціональні методи певної конкретної аплікації в один модуль, придатний для багаторазового використання.

Об'єкт запуску та закриття тестованої системи (SUT Runner). Дозволяє початковий запуск системи що підлягає тестуванню, її початкову ініціалізацію. Після закінчення тесту – звільнює ресурси пов'язані з системою.

Серед функціональних методів можна виділити набір таких, які не мають відношення до тестування функціоналу, як наприклад старт веб-браузера та навігація на логін сторінку SUT. Після закінчення тесту веб-браузер потрібно закрити. SUT Runner відповідає за такі загальні активності.

Джерело об'єктів (Object Mother). Створює об'єкти у вигляді, ініціалізованому і потрібному для виконання тесту.

Перевізник (Transporter). Централізує управління навігацією в тестованій системі в залежності від вимог тестів.

Даний об'єкт інкапсулює в собі всю логіку, пов'язану зі здійсненням навігації всередині тестованої системи. Таким чином задачі бізнес логіки не перетинається з навігацією всередині системи.

Для випадку наведеного на рисунку X, матимемо клас Transporter з методами NavigateToLogin(), NavigateToHomePage(), NavigateToCreateuser() і т.д. Як варіант, кожен окремий об'єкт сторінки може мати визначені власні методи транспортування, тоді - сам виконуватиме роль перевізника.

Композитний об'єкт сторінки (Composite Page Object). Агрегує менші об'єкти сторінки у одному зовнішньому об'єкті.

Даний патерн дозволяє структуризувати об'єкти сторінки більш об'єктно орієнтовано, виділяючи під об'єкти, що можуть перевикористовуватись на різних сторінках, і включати їх в батьківський об'єкт.

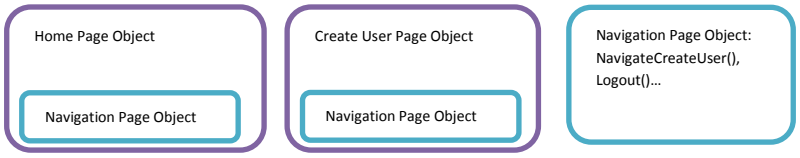

Рисунок 6 – Використання об'єкта сторінки Navigation через агрегацію в об'єктах сторінки Home та Create User

Розширений об'єкт сторінки (Extended Page Object). Розширяє базовий об'єкт сторінки через наслідування. Альтернатива композитному об'єкту сторінки.

Процесуальні патерни

Дана група патернів описує, за яким алгоритмом можна будувати функціональний тест, компонуючи його з послідовності викликів функціональних композиційних блоків.

Дано/Коли/Тоді (Given/When/Then). Ділить процес виконання тесту на три етапи:

- дано (визначення попередніх умов);
- коли (задає конкретні операції по роботі з контекстом);
- тоді (перевірка результатів).

Чотирьох етапний тест (4 stages test). Ділить процес виконання тесту на чотири етапи:

- встановлення початкових умов;
- виклик бізнес функцій;
- перевірка результатів;

- очищення системи.

Тестовий потік (Flow Test). Дозволяє на протязі одного тесту виконувати як бізнес операції так і перевірки. Вони можуть чередуватись для досягнення кінцевих цілей тесту.

Декілька помилок (Multiple Failures). Створює механізм яий дозволяє продовжувати роботу тесту після некритичної помилки.

Патерни групування тестів

Дана група описує шаблони впорядкування користувацького коду в проекті автоматизованих тестів.

Незалежний Тест (Standalone Test). Повертає тестовану систему до стану того ж в якому вона була до виконання тесту.

Пов’язаний Тест (Chained Test). Попередній тест виставляє стан тестованої системи, який потрібен буде наступним тестам.

Тестовий метод в класі (Test Method per Test Class). Окремий тестовий метод розташовується в окремому тестовому класі.

Груповані тестові методи в класі (Grouped Test Methods in Test Class). Окремий тестовий метод розташовується в окремому тестовому класі.

Висновки

В даній роботі проведено класифікацію дизайн патернів функціонального автоматизованого тестування. Вперше виділено такі патерни як Функціональний метод, Функціональна бібліотека, Об’єкт запуску та закриття тестованої системи, Композитний об’єкт сторінки, Розширений об’єкт сторінки. Визначено, що рушійною силою побудови тестувального рішення є вибір конкретного патерну імплементації тестів. Вперше виділено групи патернів функціонального автоматизованого тестування, до яких приводить багатошарова архітектура системи тестування. Зроблена класифікація патернів за їх призначенням. Вперше виділено архетип системи автоматизованого тестування у вигляді багатошарового композитного рішення з високою ізольованістю шарів та перевикористовуваним блоком – мета фреймворком.

Серед подальших активностей можна виділити наступні – порівняння патернів які є альтернативами один одному.

Список літератури

1) List of GUI testing tools Testing [Електронний ресурс] // Режим доступу до файлу: http://en.wikipedia.org/wiki/List_of_GUI_testing_tools

2) Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software/ Addison-Wesley Professional – 1994. - 416p.

3) Gerard Meszaros, xUnit Test Patterns: Refactoring Test Code / Gerard Meszaros – 2007. – 833p.

4) Misha Rybalov, Design Patterns for Customer Testing [Електронний ресурс] // Режим доступу до файлу: http://www.autotestguy.com/archives/Design%20Patterns%20for%20Customer%20Testing.pdf

5) Ryan Gerard, Amit Mathur, Meta-Framework: A New Pattern for Test Automation [Електронний ресурс]/ Режим доступу до файлу: http://www.associationforsoftwaretesting.org/?dl_name=Meta-Framework.pdf.

Відомості про авторів

Ремінний Олександр Андрійович – аспірант кафедри АІВТ. Вінницький національний технічний університет, м. Вінниця, вул. Хмельницьке шосе 95, (0432) 598243, Oleksandr.Reminnyi@gmail.com