Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

АВТОМАТИЗОВАНЕ ТЕСТУВАННЯ МОБІЛЬНИХ ЗАСТОСУВАНЬ

Текстова частина до курсової роботи

Галузь знань: 12 «Інформаційні технології»
Спеціальність: 122 «Комп’ютерні науки та інформаційні технології»

Керівник
курсової роботи
доктор технічних наук, доцент
Глибовець А.М.
(підпис)
“_—_” _____ 2020 р.

Виконав студент КНІТ-4
Єщенко М.С.
“_—_” _____ 2020 р.

Київ 2020

Тема: Автоматизоване тестування мобільних застосувань.

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	12.10.2019	
2.	Пошук тематичної літератури	20.10.2019	
3.	Ознайомлення з літературою	11.12.2019	
4.	Вивчення поняття тестування	21.12.2019	
5.	Огляд технологій для реалізації тестування в цілому	02.01.2020	
6.	Вивчення технологій для реалізації тестування застосунків для мобільних пристроїв	29.01.2020	
7.	Написання умов до програмної частини проекту	03.02.2020	
8.	Створення інфраструктури проекту для тестування	09.03.2020	
9.	Написання текстової частини курсової роботи	21.03.2020	
10.	Перегляд змісту роботи з керівником	22.03.2020	
11.	Написання аналітичної та програмної частин курсової роботи	30.03.2020	
12.	Перегляд змісту роботи з керівником	01.04.2020	
13.	Написання висновків курсової роботи	01.04.2020	
14.	Перегляд змісту роботи з керівником	02.04.2020	
15.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	03.04.2020	
16.	Створення презентації	19.04.2020	
17.	Захист роботи	24.04.2020	

Студент Єщенко М.С.

Керівник Глибовець А.М.

“____” _____

Зміст

Анотація

Метою даної роботи є дослідження методик та інфраструктурних рішень тестування програмного забезпечення для мобільних пристроїв, а також їх вплив на процес розробки й підтримки окремо взятого продукту.

У даній роботі розглядається поняття тестування, при чому розкриваються технологічні та інфраструктурні принципи створення модулів тестування для сучасних програмних продуктів, їх застосування та оцінка корисності з точки зору життєвого циклу програми.

Вступ

На перший погляд, щойно написаний код, який вдало відпрацював на комп’ютері розробника, тестування не потребує, адже усе гаразд. Проте, навіть якщо код є достатньо низькорівневим та не містить складної бізнес-логіки, він потенційно може мати неочікувану поведінку під час йог запуску із іншими вхідними даними, або ж навіть просто термінуватися під час, скажімо, його виконання на іншій операційній системі. Так, під час використання у розробці фреймворків чи SDK, таких як Flutter, код може відпрацьовувати правильно на ОС Android, але завершуватися помилкою на iOS. Що характерно, це може бути спричинено не тільки недбайливістю програміста, а й банально недопрацьованим кодом у використовуваному фреймворку. Для того щоб виправлення подібних проблем було ефективним, розроблені спеціальні комплекси та інструменти для тестування (наприклад, JUnit у Java) і навіть ідеологічні підходи до розробки програмного забезпечення, такі як TDD (test-driven development, керована тестами розробка).

Згідно «Світового звіту з якості 2019» [3], саме автоматизація тестів є вузьким місцем для забезпечення «якості на швидкості», оскільки саме автоматизація процесу тестування сприяє вдалому впровадженню методологій Agile, CI та DevOps. Наразі існують та постійно вдосконалюються рішення для автоматичного тестування усіх частин програмного комплексу – API бек-енд сервісів, графічного інтерфейсу користувача на фронт-енді, проміжних сервісів для різних задач тощо.

А отже, дослідження у галузі автоматизованого тестування є надзвичайно актуальним та корисним.

Робота складається з трьох розділів.

У першому розділі розглядається поняття тестування, для загального розуміння надається інформація про його розвиток, види, а також проводиться аналіз важливості для розробки та вплив на кінцевого рядового користувача.

Другий розділ присвячено огляду поточної ситуації на ринку та типової моделі тестування під час розробки окремо взятої компанії певного продукту. Це є важливим для формування розуміння ситуації на сьогодні – наявних можливостей та відомих проблем, які ще не розв’язала ані наука, ані індустрія. На цю інформацію спирається й практична частина роботи.

Третій розділ, є спробою показати сучасне комплексне тестування на практиці, а також передбачити можливий подальший рух на думку автора, який привів би до покращення в галузі автоматизованого тестування.

Постановка задачі.

1. Пояснити та порівняти підходи до автоматизації тестування програмних продуктів й надати інформацію про актуальність їх використання на сьогодні.
2. Виконати аналіз поточного стану ринку, набравши досвід для виконання автоматизованого тестування та запропонування пропозицій для подальшого розвитку предметної області.
3. Розробити комплексне автоматизоване тестування програмного продукту, вказавши шляхи поліпшення існуючого інструментарію та підходів до його використання.

Розділ 1: Вступ до тестування програмного забезпечення

1.1 Поняття тестування програмного забезпечення

Тестування програмного забезпечення – це процес технічного дослідження, призначений для виявлення інформації про якість продукту відносно контексту, в якому він має використовуватись, а також визначення відповідності вимогам суб'єкта розробки (на практиці часто замовника, який може бути як фізичною, так і юридичною особою, і при цьому не обов'язково мати щось окрім базових знань у предметній області продукту). Техніка тестування також включає як процес пошуку помилок або інших дефектів, так і випробування програмних складових з метою умовної оцінки. Здійснюється як шляхом спостереження за його роботою в штучно створених спеціалістом ситуаціях, так і автоматизовано, на обмеженому наборі тестів, згенерованих певним чином; часто у такому режимі головною метою є перевірка виконуваності різних інваріантів, що надає змогу відстежити загальну коректність логіки програми під час виконання її різноманітних модулів, підпроцедур чи інших частин, або ж і її загалом [4].

Критерії оцінки тестування:

a) відповідність технічному завданню, що використовувалося під час розробки продукту архітекторами та програмістами;
b) коректність роботи на усіх можливих вхідних даних;
c) коректне завершення програми за прийнятний час;
d) практичність;
e) сумісність із операційними системами, вказаними у технічному завданні;
f) відповідність баченню та потребам замовника, у першу чергу таким, які були формалізовані та задокументовані у технічному завданні.

Час на тестування відводиться в залежності від обраного підходу до розробки програмного забезпечення. Так, у режимі розробки «водоспад»,

тестування є окремою фазою, яка є наступним етапом після побудови продукту. А ось у випадку розробки за методологією «Agile» («спритної»), тестування є інтегрованою, часто безперервною задачею, яка проводиться паралельно із розробкою та впровадженням усіх модулів.

1.2 Коротка історія розвитку тестування програмного забезпечення

Ще з 1960-х рр. потреба розроблювати надійні та ефективні застосунки породила процедуру тестування. На той момент воно здебільшого представляло з себе інспекцію коду та перевірку на працездатність із різноманітними наборами вхідних даних.

У 1970-х рр. тестування стало значно більш розповсюдженим, як і ОЕМ, на яких писалися програми. Це зсунуло парадигму тестування далі: «тестування – це процес, спрямований на демонстрацію коректності продукту» [4].

1980-і рр. стали початком написання превентивних тестів. Це значно підвищило ефективність процедури та уможливило ітеративний підхід до розробки програмного забезпечення. Тоді ж вперше виникло автоматизоване тестування, оскільки комп’ютер міг перевірити більшу кількість тестів за одиницю часу, ніж людина-тестувальник.

У 1990-х рр. автоматизація вийшла на новий рівень, дозволивши генерувати звіти після виконання тестів; виникло спеціалізоване програмне забезпечення для написання та виконання тестів; а саме поняття тестування почало також включати в себе планування, проектування, створення та підтримку тестів.

Останні дві декади ознаменували підтримку тестуванням методології «Agile», появу тестів «у хмарі», DevOps, CI/CD.

У найближчому майбутньому ймовірне розповсюдження технік машинного навчання та штучного інтелекту на створення комплексів тестів та оптимізацію їх виконання. Враховуючи глобалізацію та кількість одночасних користувачів, а

також надзвичайно широкі можливості сучасних застосунків (а інколи й велику кількість коду, написаного протягом багатьох років), інтелектуальний підхід до автоматизації тестування є вкрай актуальним.

1.3 Методи тестування програмного забезпечення

Існує багато ефективних методів тестування програмного забезпечення – метою цього підрозділу є ознайомлення із найбільш важливими з них, які відіграють значну роль сьогодні і, скоріше за все, ще довго залишатимуться актуальними та перспективними.

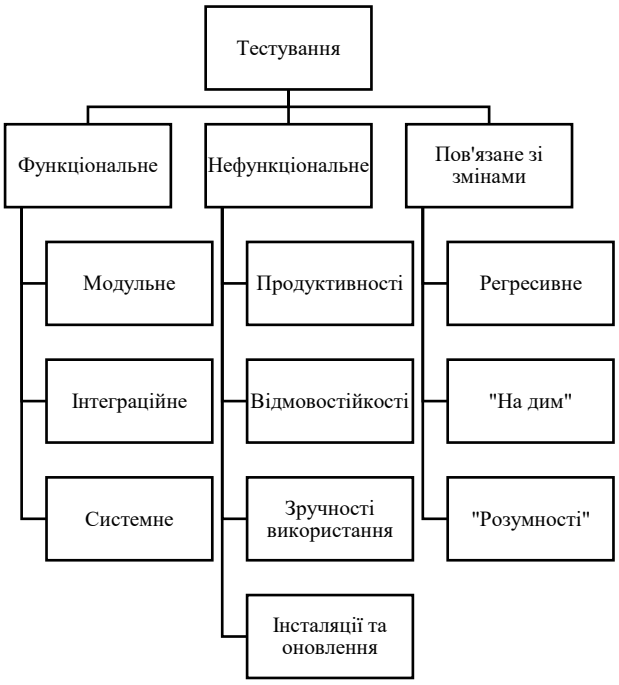

Рисунок 1 – Методи тестування

З діаграми видно, що методи тестування поділяються на три основні категорії: функціональне, нефункціональне та таке, що безпосередньо пов’язане зі змінами.

Іншими словами, функціональне тестування направлене на перевірку коректності розв’язання продуктом поставлених перед ним задач. Серед вимог

може бути відповідність коду та його логіки стандартам, захищеність, точність і т.ін.

Три найбільших категорії такого тестування є модульне (тестування окремих частин коду, які можна вважати достатньо «самостійними» для проведення певного тесту), інтеграційне (перевірка взаємодії певного коду з іншими частинами продукту) та системне (найбільш загальне тестування продукту, направлене на перевірку програми в цілому – за кількістю використовуваних ресурсів та продуктивністю у різних режимах використання, у т.ч. у випадку високого навантаження («стрес-тест») і т.д.).

Нарешті, пов’язане зі змінами тестування, що поділяється на регресивне, «на дим» та на «розумності» потребує деяких пояснень. Регресивне дозволяє виявити помилки у тих частинах коду, які були створені до внесення останніх змін та виявити проблеми у тому коді, де все мало б працювати правильно, як і до нового патчу. «На дим» покликано виявити найбільш очевидні помилки, та у разі їх появи – розпочати більш глибинне тестування. «Розумності» - тестування для перевірки базової логіки коду, яка може у деяких випадках суперечити логіці розв’язання поставленої перед програмою задачі.

Нефункціональне тестування можна умовно поділити на такі чотири найбільш вагомі категорії: продуктивності, відмовостійкості, зручності використання й інсталяції та оновлення. Сенс даних видів тестів зрозумілий з їх назв і є надзвичайно важливим для створення повноцінного продукту.

Окрім цього, тестування також може класифікуватися за ознаками. Так, найбільш важливою категоризацією є «за ступенем автоматизації» із такими можливими видами:

a) ручне;
b) напівавтоматизоване;
c) автоматизоване.

Ручне є найбільш примітивним, але часто необхідним для наближення процесу перевірки до моделі використання програмного продукту кінцевим користувачем. Автоматизоване є таким, яке дозволяє виконати заздалегідь написані тести для перевірки коректності роботи застосунку та проаналізувати результат виконання. Напівавтоматизоване поєднує ці дві крайнощі, так як цього вимагає предметна область.

Серед інших класифікацій – за етапом готовності продукту (альфа-, бета-тестування), за ступенем підготовленості до тестування (за документацією чи інтуїтивне) і т.ін.

1.4 Перебіг тестування

Типовий процес тестування під час виходу кожної нової версії програмного забезпечення умовно можна поділити на шість етапів:

1. Розподіл пріоритетності тестування відносно різних програмних модулів (у випадку продукту, орієнтованого на велику кількість зовнішніх користувачів, багато уваги відводиться саме тим модулям, до яких у користувачів є відкритий доступ, оскільки вони і є найбільш вразливими з точки зору безпеки та критичними з точки зору використання їх кінцевими користувачами програми).
2. Прийняття рішення стосовно набору тестів, які обов’язково має пройти певний програмний модуль задля перевірки коректності та надійності його роботи.
3. Логування результатів (є надзвичайно важливим задля пост-аналізу проходження тестів; окрім вдало/невдало також логуються додаткові свідчення, такі як виниклі помилки, які в подальшому значно допомагають у виправленні помилок в коді).

4. Визначення пріоритетних програмних помилок за серйозністю проблеми та швидкістю її виправлення через аналіз логів.
5. Виправлення найбільш критичних багів.
6. Збір даних та збереження у трекері помилок із меншим пріоритетом задля подальшого їх виправлення.

Обрання інструментів та методик для тестування у даному випадку є метаінформацією, оскільки вона може дуже відрізнятися в залежності від конкретної предметної області.

1.5 Інші важливі поняття

Для повноцінного розуміння процесу та важливості тестування програмного забезпечення, нижче наведено перелік важливих понять, які безпосередньо стосуються теми.

Покриття коду – міра кількості протестованого коду програми відносно усього коду, визначається у відсотках. Так, процедура вважається покритою у разі, якщо хоч один зі шляхів виходу з неї був досягнений під час тестування. 100% покриття у даному випадку досягається тоді, коли були досягнені усі можливі виходи з даної процедури. Окрім процедур, вимірюється також покриття операторів, умов, викликів.

Приймальне тестування – процедура, яка існує задля затвердження факту працездатності та відповідності програми технічному завданню. Виконується під час здачі програмного продукту замовнику. Результатом є рішення стосовно прийняття продукту та винесення висновків стосовно стану його готовності на практиці.

DevOps – методологія розробки програмного забезпечення, що сприяє ефективній та постійній взаємодії усіх відділів, задіяних у процесі розробки, тестування та підтримки продукту. Інструментарій включає у себе засоби

безперервної розробки та інтеграції, контейнеризації, віртуалізації та й інші, за потребою.

CI/CD (неперервна інтеграція/розробка) – один з інструментів DevOps, практика розробки програмного забезпечення, що пропагує автоматизовану інтеграцію коду, написаного незалежними командами в рамках роботи над спільним проектом. Важливою частиною процесу є виконання автоматизованого тестування після вдалої побудови проекту. Наразі саме цей напрямок і є найбільш прогресивним та перспективним для розвитку автоматизованого тестування, особливо якщо взяти до уваги ітеративність процесу (перебудова проекту може здійснюватися після кожного коміту членів команди) та глибоку інтегрованість у загальний перебіг розробки.

Розділ 2: Тестування програмного забезпечення сьогодні

2.1 Місце тестування програмного забезпечення у контролі якості підприємства

Найбільш широким поняттям є Quality Assurance (QA, запевнення у якості продукту відносно вимог). Воно включає у себе зокрема Quality Control (QC, контроль за якістю шляхом виявлення помилок), що має вже певні конкретні регламентовані процедури, серед яких і є тестування програмного забезпечення.

Рисунок 2 – Місце тестування у контролі якості підприємства

2.2 Сучасні аспекти тестування програмного забезпечення

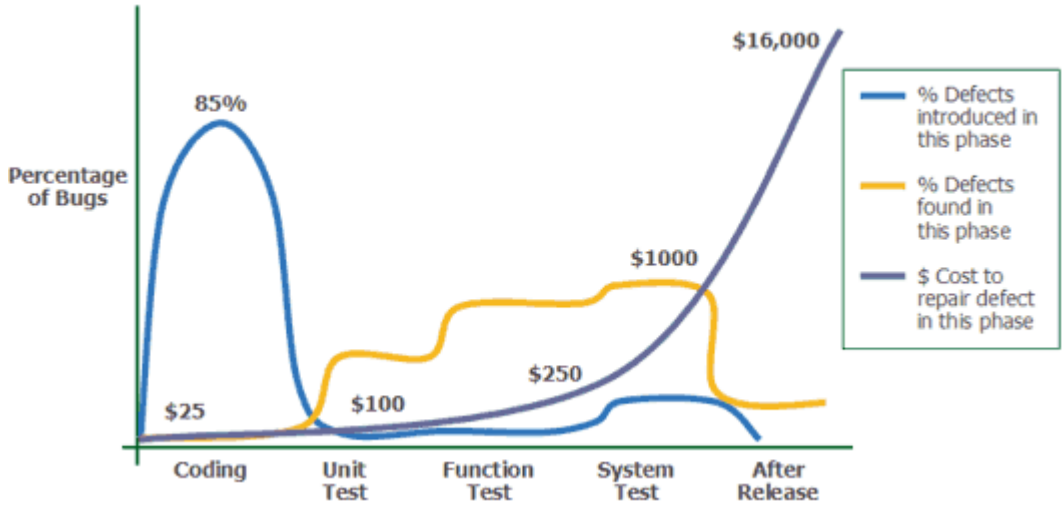

Рисунок 3 – Вартість виправлення помилки у коді [7]

Незважаючи на дату публікації, з якої взяте зображення вище, можна сміливо стверджувати, що на сьогодні ситуація майже не змінилася. Це підтверджують більш сучасні джерела, такі як [8]. Розвиток та адаптація методології «Agile», звісно, приніс свої плоди, але це мало як допоможе у випадку, коли розробкою продукту займається одна команда, а підтримкою – інша, яку рекрутують ситуативно задля скорочення коштів на утримання штату. Такий приклад розподілу ролей яскраво ілюструє неможливість швидко розібратися у виниклій проблемі. Задля покращення швидкості реагування необхідно більше людських ресурсів, а отже й грошей. Саме тому підходити до процесу тестування необхідно якомога більш ретельно та відповідально.

2.3 Автоматизоване тестування та ринок мобільних застосунків, ситуація сьогодні та прогноз розвитку

Перехід різних галузей промисловості до мобільних робить мобільний сегмент одним з найбільш перспективних серед споживчих у IT-сфері. Згідно з дослідженням «Worldwide Semiannual Mobility Spending Guide»,

проведеним корпорацією IDC (International Data Corporation), очікується, що витрати на мобільні рішення сягнуть 1.72 трлн у 2021 році [9]. Через стрімкий розвиток мобільного ринку стверджувати, що він повністю готовий підтримувати усі складні архітектурні рішення, які давно існують на більш традиційних платформах, неможливо. Виникають складнощі із оптимізацією та гарантуванням коректної роботи на різних пристроях. Проблема автоматизованого тестування стає актуальною як ніколи до цього.

Згідно докладу Transparency Market Research, глобальний ринок автоматизованого тестування розшириться на 15,4% з 2017 до 2025 року, загальна вартість зросте до $109,69 млрд у 2025 році з $15,87 млрд у 2016 [9].

Головними мобільними операційними системами на ринку є Android та iOS. Оскільки Android займає більшу частину ринку, робота буде сконцентрована саме навколо нього.

Для Android наразі існує безліч інструментів для автоматизації тестування, багато з них є кроссплатформенними. Серед них (тестування клієнтської частини): Appium, Robotium, UI Automator, XCUITest, Eggplant, Ranorex та інші. Postman, RestAssured, SoapUI, Fiddler – для автоматизації тестування серверної частини. Існують й інші, більш вузькоспеціалізовані утиліти для автоматизації тестування певних частин проектів.

Наразі на ринку мобільних застосувань продовжують з’являтися усе нові технології та усе нові пристрої, що ускладнює процес тестування та збільшує кількість необхідних тестів. На допомогу приходять засоби автоматизації тестування, що використовують машинне навчання та штучний інтелект. Тож саме розвиток науки у цих галузях має допомогти розв’язати проблему необхідності тестувати все більше різного коду.

Розділ 3: Тестування мобільного застосунку на ОС Android

3.1 Предметна область та обрані інструменти автоматизації тестування

Обраною предметною областю став проект плеєру медійного контенту. На цьому прикладі можна показати тестування різних компонентів типового архітектурно складного застосунку. Протягом існування даного застосунку використовувалися різні засоби для автоматизації тестування як фронт-енду, так і використовуваних API, але сучасний підхід використовує Espresso та Postman.

Типовий «hello world» тест фронт-енду на мові Java, використовуючи Espresso виглядає так:

```
@Test
public void testHelloWorld() {
    onView(withId(R.id.name_field)).perform(typeText("Test_1"));
    onView(withId(R.id.proceed_button)).perform(click());
    onView(withText("Hello Test_1!")).check(matches(isDisplayed()));
}
```

Більш складними тестами можуть бути такі:

-Тест на зміну тексту.

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class SwitchLabelTest {

    public static final String TEST_TEXT = "Espresso";


    @Rule public ActivityScenarioRule<MainActivity> activityScenarioRule
            = new ActivityScenarioRule<>(MainActivity.class);

    @Test
    public void switchLabel_sameActivity() {
        onView(withId(R.id.switchLabelIn)).perform(typeText(TEST_TEXT),
closeSoftKeyboard());
        onView(withId(R.id.switchLabel)).perform(click());
```

```
onView(withId(R.id.switchLabelText)).check(matches(withText(TEST_TEXT)));
          }

          @Test
          public void switchLabel_newActivity() {
              onView(withId(R.id.switchLabelIn)).perform(typeText(TEST_TEXT),
closeSoftKeyboard());
              onView(withId(R.id.switchLabelTextBt)).perform(click());

onView(withId(R.id.show_text_view)).check(matches(withText(TEST_TEXT)));
          }
      }
```

-Тест відображення веб-сторінки:

```
      import org.junit.Before;
      import org.junit.Rule;
      import org.junit.Test;
      import org.junit.runner.RunWith;

      import android.content.Intent;

      import androidx.test.core.app.ActivityScenario;
      import androidx.test.espresso.web.sugar.Web;
      import androidx.test.espresso.web.webdriver.DriverAtoms;
      import androidx.test.espresso.web.webdriver.Locator;
      import androidx.test.filters.LargeTest;
      import androidx.test.rule.ActivityTestRule;
      import androidx.test.ext.junit.runners.AndroidJUnit4;
      import android.webkit.WebView;

      import static
androidx.test.espresso.web.assertion.WebViewAssertions.webMatches;
      import static androidx.test.espresso.web.sugar.Web.onWebView;
      import static androidx.test.espresso.web.webdriver.DriverAtoms.clearElement;
      import static androidx.test.espresso.web.webdriver.DriverAtoms.findElement;
      import static androidx.test.espresso.web.webdriver.DriverAtoms.getText;
      import static androidx.test.espresso.web.webdriver.DriverAtoms.webClick;
```

```
import static org.hamcrest.Matchers.containsString;


@LargeTest
@RunWith(AndroidJUnit4.class)
public class WebViewTest {

    private static final String KeyOne = "key_one";
    private static final String KeyTwo = "key_two";

    @Rule
    public ActivityTestRule<WebViewActivity> mActivityRule = new
ActivityTestRule<WebViewActivity>(
            WebViewActivity.class, false, false) {
        @Override
        protected void afterActivityLaunched() {
            onWebView().forceJavascriptEnabled();
        }
    };

@Test
    public void typeTextInInput_clickButton_ChangesText() {
        mActivityRule.launchActivity(withWebFormIntent());
        onWebView()
                .withElement(findElement(Locator.ID, "text_input"))
                .perform(clearElement())
                .perform(DriverAtoms.webKeys(KeyTwo))
                .withElement(findElement(Locator.ID, "changeTextBtn"))
                .perform(webClick())
                .withElement(findElement(Locator.ID, "message"))
                .check(webMatches(getText(), containsString(KeyTwo)));
    }

    private static Intent withWebFormIntent() {
        Intent basicFormIntent = new Intent();
        basicFormIntent.putExtra(WebViewActivity.KEY_URL_TO_LOAD,
WebViewActivity.WEB_FORM_URL);
        return basicFormIntent;
```

```
    }

    @Test
    public void typeTextInInput_clickButton_SubmitsForm() {
        mActivityRule.launchActivity(withWebFormIntent());
        onWebView()
                .withElement(findElement(Locator.ID, "text_input"))
                .perform(clearElement())
                .perform(DriverAtoms.webKeys(KeyOne))
                .withElement(findElement(Locator.ID, "submitBtn"))
                .perform(webClick())
                .withElement(findElement(Locator.ID, "response"))
                .check(webMatches(getText(), containsString(KeyOne)));
    }


}
```

Висновки

Задача автоматизації тестування та створення ефективних підходів до автоматизованого тестування, особливо на ринку мобільних застосувань, є надзвичайно актуальною та потребує все більшого вивчення. На даному етапі наука покладається на методи машинного навчання та штучний інтелект як новітні інструменти швидкої та ефективної генерації тестів, проходження яких дозволило б дати точну відповідь на питання, чи працює програмний код коректно та чи відповідає його робота закладеній у нього логіці.

Підсумовуючи, можна сказати що існуюча динаміка пошуку сучасних рішень розв’язання задачі автоматизації тестування є надзвичайно корисною й для інших галузей розробки програмного забезпечення, а також і для кінцевого користувача, оскільки постійно створюються все більш досконалі й водночас універсальні інструменти для засвідчення у якості та досконалості виконання програмного коду.

Список використаної літератури

1. https://usersnap.com/blog/software-testing-basics/
2. https://en.wikipedia.org/wiki/Test-driven_development
3. https://www.sogeti.com/explore/reports/world-quality-report-2019
4. https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F
5. https://www.slideshare.net/eleksdev/13-testing
6. https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/
7. *Applied Software Measurement*, Capers Jones, 1996
8. http://old.semat.org/wp-content/uploads/2012/10/Capers_Jones.pdf
9. Українське дослідження глобального ринку автоматизації мобільного тестування, його проблем та перспектив за 2018 рік, здійснене компанією QATestLab: https://qatestlab.com/assets/Uploads/Whitepaper-Future-of-Mobile-Test-Automation.pdf?__s=ibzcg94llhx0n2waq7be