**Лекція 20 Багтрекінг. Інструменти для автоматизації тестування**

*Тестування ПЗ* — це процес перевірки програмного продукту з метою виявлення дефектів, помилок та недоліків перед його випуском на ринок або в експлуатацію. Цей процес охоплює запуск програми з різними вхідними даними та умовами, а також аналіз реакції програми на ці дані. *Мета тестування* — це підтвердження правильності роботи програми відповідно до вимог до неї, а також забезпечення високої якості та надійності програмного продукту.

## Методи тестування програмного забезпечення

Тестування програмного забезпечення може проводитися як вручну, так і за допомогою автоматизації. Обидва підходи мають свої переваги та недоліки, які варто враховувати при плануванні тестових стратегій. Розгляньмо коротко основні аспекти кожного з них.

### Ручне тестування

Ручне тестування (Manual testing) – це набагато більше, ніж просто проведення тестів. Це спосіб пошуку вразливостей та свідомого виявлення непомічених раніше дефектів, у якому людський фактор є основним додатковим засобом для впевненості в якості продукту. У світі багаторівневого програмного забезпечення ручне тестування є важливим етапом, який розкриває те, що запрограмовані тести не можуть. Це процес, у якому відображається мистецтво та вміння виявляти помилки, які інші методи можливо пропустили б.

#### Переваги

* Гнучкість та можливість адаптації до змін
* Ефективне для випадків тестування, які складно автоматизувати
* Легке виявлення нових дефектів, які не були враховані в автоматизованих скриптах

#### Недоліки

* Велика витрата часу та ресурсів, особливо на великих проєктах
* Підвищений ризик людської помилки, який може призвести до пропуску деяких дефектів або до неправильної оцінки їх важливості
* Повторне виконання однотипних тестів може призвести до втоми, що впливає на якість тестування

Тим не менш, перед тим як автоматизувати тестування будь-якого додатку, необхідно спочатку виконати серію тестів вручну. Мануальне тестування вимагає більших зусиль, але без нього ми не зможемо переконатися в тому, чи можлива автоматизація взагалі. Один із фундаментальних принципів тестування свідчить: **100% автоматизація неможлива**. Тому, ручне тестування – це необхідність.

**Міфи про ручне тестування:**

* – Хто завгодно може провести ручне тестування.

Ні, виконання будь-якого виду тестування вимагає спеціальних знань та професійної підготовки.

* – Автоматизоване тестування краще, ніж ручне.

Повна автоматизація неможлива. Необхідно використовувати також і ручне тестування.

* – Ручне тестування – це просто.

Тестування може бути дуже непростим заняттям. Проведення тестування для перевірки максимально можливої кількості шляхів виконання програми із використанням мінімального числа тест-кейсів вимагає серйозних аналітичних навичок.

### Приклад тестування графічного інтерфейсу

### Вичерпний список перевірок, які необхідно здійснити в рамках тестування графічного інтерфейсу і на що треба звертати увагу при кожній з них, виглядає наступним чином:

### - загальний вигляд сторінки: перевірка цілісності зовнішнього відображення на відповідність дизайну та макетам, загальної гармонії, реакції продукту на масштабування, тощо;

### - текст: перевірка тексту на наявність помилок та похибок, правильне форматування та однакове вирівнювання в усіх місцях продукту;

### - вибір елементів: перевірка інтуїтивно зрозумілого виділення елементів, з якими кінцевий користувач може взаємодіяти на різних етапах цієї взаємодії (наприклад, підсвічення тексту з посиланням яскравим кольором на стартовій сторінці, виведення маленького віконця з додатковою інформацією при наведенні на посилання і відкриття повноцінного ресурсу після кліку на посилання);

### - робота з полями: перевірка того, що при загальній уніфікації полів, статичні поля, які не можна редагувати, чітко відрізняються від полів, які потребують від кінцевого користувача введення інформації;

### - перевірка зовнішнього вигляду та розташування форми, а також її наповнення: радіокнопок, чекбоксів, випадаючих списків, кнопок, текстових полей тощо;

### - інші вимоги: специфічні вимоги, повʼязані з замовником чи середовищем таким як маркетплейс (онлайн-ринок електронної комерції), в якого можуть бути свої специфікації, яких необхідно дотримуватись, інакше продукт може не пройти перевірку та не бути розміщеним на бажаних ресурсах.

### Автоматичне тестування

Автоматичне тестування (Automation testing) – це процес виконання тестів та перевірки результатів за допомогою автоматизованих засобів, таких як тести без користувача, скрипти тестування, програмні платформи тощо. Цей метод стає все більш важливим у сучасній розробці програмного забезпечення, оскільки дозволяє ефективно автоматизувати та прискорити процес тестування, зменшуючи людський вплив та забезпечуючи послідовність та швидкість виконання тестів.

#### Переваги

* Швидке виконання великої кількості тестів
* Забезпечення постійної та повторюваної перевірки продукту
* Допомагає у виявленні дефектів на ранніх етапах розробки
* Автоматичне виконання регресійних тестів на великому масштабі, що допомагає уникнути деградації функціональності в процесі розширення системи

#### Недоліки

* Потребує часу та зусиль на розробку та підтримку автоматизованих тестів
* Деякі аспекти, такі як візуальна перевірка та сприйняття, може бути складно автоматизувати
* Автоматичні тести вимагають постійної підтримки та оновлення під час змін у програмному забезпеченні

**Автоматизоване тестування** припускає використання спеціального програмного забезпечення (окрім того, що тестується) для контролю виконання тестів та порівняння очікуваного і фактичного результату роботи програми. Цей тип тестування допомагає автоматизувати дії, що часто повторюються, але які, в той же час, необхідні для максимального тестового покриття завдання.

Деякі завдання тестування, такі як – низькорівневе регресійне тестування, можуть бути трудоємкими та вимагати багато часу, якщо їх виконувати вручну. Окрім того, мануальне тестування може недостатньо ефективно знаходити деякі класи помилок. У таких випадках автоматизація може допомогти заощадити час і зусилля проектної команди.

Після створення автоматизованих тестів, їх можна в будь-який момент запустити знову, причому запускаються і виконуються вони швидко та точно. Таким чином, якщо є необхідність частого повторного прогону тестів, значення автоматизації для спрощення супроводу проекту і зниження його вартості важко переоцінити. Адже навіть мінімальні патчі та зміни коду можуть стати причиною появи нових багів.

**Існує кілька основних видів автоматизованого тестування:**

– Автоматизація тестування коду (Code-driven testing) – тестування на рівні програмних модулів, класів та бібліотек (фактично, автоматичні юніт-тести);

– Автоматизація тестування графічного інтерфейсу користувача (Graphical user interface testing) – спеціальна програма (фреймворк автоматизації тестування) дозволяє генерувати користувальницькі дії – натиснення кнопок, кліки мишкою, та відслідковувати реакцію програми на ці дії – чи відповідає вона специфікації.

– Автоматизація тестування API (Application Programming Interface) – програмного інтерфейсу програми. Тестуються інтерфейси, призначені для взаємодії, наприклад, з іншими програмами або з користувачем. Тут, знову ж таки, як правило, використовуються спеціальні фреймворки.

Для складання автоматизованих тестів, QA-фахівець повинен вміти програмувати. *Автоматичні тести – це повноцінні програми, просто призначені для тестування.*

Коли, що і як автоматизувати і чи автоматизувати взагалі – дуже важливі питання, відповіді на які повинна дати команда розробки. Вибір правильних елементів програми для автоматизації більшою мірою визначатиме успіх автоматизації тестування в принципі. Потрібно уникати автоматизації тестування ділянок коду, які можуть часто змінюватися.

**Порівняння ручного та автоматизованого тестування**

Як ручне, так і автоматизоване тестування можуть використовуватися на різних рівнях тестування, а також бути частиною інших типів і видів тестування.

**Автоматизація зберігає час, сили та гроші.** Одного разу автоматизований тест можна запускати знову і знову, докладаючи мінімум зусиль.

**Вручну можна протестувати практично будь-який додаток, в той час як автоматизувати варто тільки стабільні системи.** Автоматизоване тестування використовується головним чином для регресії. Крім того, деякі види тестування, наприклад, ad-hoc або дослідницьке тестування можуть бути виконані тільки вручну.

**Мануальне тестування може бути повторюваним та нудним.** У той же час, автоматизація може допомогти цього уникнути – за вас все зробить комп’ютер.

Таким чином, на реальних проектах найчастіше використовується комбінація ручного і автоматизованого тестування, причому рівень автоматизації буде залежати як від типу проекту, так і від особливостей постановки виробничих процесів в компанії.

Якісна автоматизація процесу тестування дозволяє не лише втілити всі наведені вище переваги перевірки користувацького інтерфейсу, а й в значній мірі пришвидшити його, адже з впровадженням практик безперервної інтеграції та безперервної доставки (Continuous integration / Continuous delivery, CI/CD) участь людини не знадобиться не лише в процесі безпосереднього виконання тесту, а й навіть для його запуску, адже він починатиметься автоматично як тільки певна частина коду буде зачеплена, і по завершенні, бо вивантаження результатів прогону тестів можна також налаштувати автоматично.

Крім того, з залученням автоматизації, можна уникнути помилок, повʼязаних з людським фактором по типу неуважності чи втоми, або нестачі робочого часу, адже середовище виконання тесту не залежить від таких обставин.

Разом з тим, необхідно також розуміти, що автоматизація рутинних процесів тестування користувацького інтерфейсу може мати і певні обмеження, як, наприклад, відсутність людської взаємодії, критичного мислення та креативного підходу у випадках, коли треба перевірити нестандартну поведінку програми, або неефективна витрата часу на налаштування емуляцій для точкової перевірки невеличкої правки в коді. Також не слід забувати, що саме при ручному тестуванні створюється найбільш приближений до реального контекст для пошуку проблем, на які зазвичай потрапляють кінцеві користувачі, адже реальну поведінку людини не так легко симулювати і тому деякі дефекти можуть бути пропущеними чи непоміченими під час автоматизованого тестування графічного користувацького інтерфейсу.

**Шляхи уникнення обмежень, які накладаються на автоматизацію тестування користувацького інтерфейсу**.

Логічним буде припустити, що для превалювання переваг автоматизованого тестування користувацького інтерфейсу над його обмеженнями достатньо вибудувати правильну стратегію покриття функціоналу автоматизованими тестами та мати реалістичні очікування в їх застосуванні. Так, зазвичай, автоматизації піддаються прості позитивні перевірки, тобто поведінка, яку ми очікуємо отримати в безумовній більшості випадків, тоді як негативні сценарії та нетипова (унікальна) поведінка найкраще перевіряється вручну.

Найбільш розповсюджені підводні камені UI-тестування та шляхи їх вирішення наведені нижче.

1) «тестування займає багато часу»:

- запуск тестів на паралельне виконання або з використанням більше ніж одного віртуального середовища;

- групування тестів на основі передумов та почерговий запуск тестів без необхідності відтворювати середовище з нуля;

- виконання базового сценарію в першу чергу, виключаючи необхідність тестування комплексного функціоналу у випадку, якщо зламалося щось основоположне;

- нічний запуск тестів;

- першочергова перевірка кількох Е2Е-послідовностей (end-to-end або наскрізне тестування, що перевіряє продукт від початку до кінця);

- підтримання атомарності окремих тестів задля забезпечення можливості їх окремого запуску без необхідності запускати цілий тестовий набір;

- перенесення деяких передумов або перевірка кінцевих результатів з UI на бекенд. Наприклад, якщо для передумови тесту потрібне створення нового користувача і це не є основною перевіркою сценарію, його можна створити на бекенді;

2) нестабільні/динамічні елементи:

- встановлення заборони або обмеження на оновлення середовища під час роботи тесту;

- замість неявного очікування, яке прописується в одиницях часу, використовувати явне очікування на взаємодію з необхідним елементом;

- застосування повторних спроб підтвердження наявності необхідного елементу, в помірній кількості;

- логування результатів запуску у вигляді наочного матеріалу: скріншотів, гіф, коротких відео;

3) «тести не помічають зміни в UI»:

- застосування наявних або створення власних додаткових інструментів для порівняння скріншотів з головних сторінок, вкладок, пунктів, тощо. Це підвищить вірогідність того, що зміни, які не було виявлено тестами, будуть «відловлені» за допомогою порівнянь.

**Підходи до тестування UI у web-сервісах: record-and-playback та тестування на основі моделей**

При виборі підходів до автоматизації тестування графічного користувацького інтерфейсу, першими на думку спадають два найбільш поширені: «запис-та-відтворення» (record-and-playback) і тестування на основі моделей (model-based testing).

Основа методу запису-та-відтворення лежить в його назві це запис усіх взаємодій користувача з web-застосунком та їх відтворення в тій самій послідовності за допомогою спеціальних скриптів. Виконання таких скриптів повторюється знову і знову без втручання тестувальника або іншого зацікавленого члена команди. Такий спосіб підходить, зокрема, для простих інтерфейсів з невеликою кількістю функцій. З плюсів підходу record-and-playback відразу можна відмітити легкість у використанні та відсутність необхідності у тестувальника знань та вмінь писати код.

Звичайно, є і свої мінуси. Даний підхід вимагає постійного контролю: скрипти можуть не відтворюватися коректно у разі незначних оновлень, переміщення UI-елементів, використання нових ID або класів. Також, іноді простіше та швидше перевірити помилку вручну на вже наявному наборі даних, аніж перезаписувати весь скрипт з самого початку чи якоїсь конкретної відправної точки. Але, безперечно, все залежить від інструментів, які використовує команда розробки, та функціональності, яка перевіряється.

Розширене порівняння переваг та недоліків підходу наведене у таблиці 1.

Таблиця 1 - Переваги та недоліки record-and-playback підходу автоматизації тестування

|  |  |
| --- | --- |
| Переваги | Недоліки |
| Відсутність необхідності знань та вмінь писати код у тестувальника. | Зміни в інтерфейсі продукту вимагають перезапису тестових сценаріїв з нуля. |
| Будь-який зацікавлений член команди розробки зможе створити свої тести після засвоєння роботи з обраним інструментом. | Труднощі підтримки автотестів при великій їх кількості, що забирає час та ресурс у тестувальника. |
| Підхід забезпечує легкий перехід з ручного тестування в автоматизоване. | Відсутність необхідності вміння кодувати для створення таких автотестів забирає варіативність. |

Підхід захоплення-та-відтворення ідеально підходить для невеликих проектів або стартапів з незначною кількістю тестів, тоді як для великих або стрімко зростаючих проектів з високим покриттям функціональності він може виявитись занадто «неповоротким».

Прикладами інструментів, що пропонують зазначений підхід на сучасному

ринку, є:

- Katalon;

- Selenium IDE;

- TestComplete;

- Testim;

- Ranorex Studio;

- Telerik UI.

Тестування на основі моделей, в свою чергу, більше підходить для сценаріїв, коли інженери створюють абстрактну модель з описом функціональності та будують процес тестування на її основі. Проте такий спосіб вимагає певних знань і досвіду в автоматизації та програмуванні. Написання оптимального під потреби проекту та команди фреймворку для тестування займає певний час і для тестувальників, які слабо розуміються на технічних питаннях, «прочитати» та зрозуміти всі взаємозв’язки та залежності може бути важко. Однак цю та інші складнощі більшою мірою досить просто обійти завдяки рішенням, що пропонують сучасні інструменти, тому вважатимемо, що даний метод має більше переваг, ніж вищезазначений record-and-playback.

Тестування на основі моделей заощаджує час та гроші, адже його легко підлаштувати під різні умови: середовища, машини, браузери та інше. Якщо знадобиться покрити нові функції, зазвичай це не спричинить таких додаткових витрат, щ обули описані вище, як у тестуванні методом запису-та-відтворення. І останнє, але не менш важливе: завдяки методу на основі моделей легко виявляти та виправляти помилки.

Разом з тим необхідно памʼятати про те, що тестування на основі моделей має бути виконано в чітко визначених рамках:

1) моделювання - етап збору та опису вимог, які будуть використовуватись для створення тестів. На цьому ж етапі передбачається окреслення очікуваного результату. Даний етап відповідає на питання «Що буде тестуватися?»;

2) планування тестування - етап визначення критеріїв тестування, відповідає на питання «Як буде тестуватися функціонал?»;

3) тест-дизайн - етап вибору підходу до тестування на основі попередньо зіставлених критеріїв, відповідає на питання «Яким чином буде тестуватися функціонал?»;

4) створення тестової документації (тест-кейсів) - етап безпосереднього написання вичерпних тестових сценаріїв;

5) виконання тестування - етап безпосереднього виконання тестування і порівняння очікуваного результату з фактичним.

**Автоматизоване тестування https://qalight.ua/baza-znaniy/avtomatizovane-testuvannya/**

**Автоматизоване тестування** програмного забезпечення є **важливим** з наступних причин:

* Ручне тестування усіх робочих процесів, усіх полів, усіх негативних сценаріїв вимагає багато часу та грошей.
* Доволі складно протестувати мультимовні сайти вручну.
* Автоматизація не вимагає втручання людини. Ви можете запустити автоматичний тест без нагляду (наприклад вночі).
* Автоматизація збільшує швидкість виконання тесту.
* Автоматизація допомагає збільшити покриття тестами (Test Coverage).
* Ручне тестування може бути нудним а, отже, веде до випадкових помилок.

*Умовно, автоматизоване тестування можна поділити на дві частини*:

1. Пошук елемента за локатором
2. Дії з виявленим елементом.

У веб-документі ми шукатимемо за DOM структурою, а саме за X-Path локатором або CSS-локаторами. На desktop і mobile пошук найчастіше здійснюється за своїми координатами. А після пошуку виконується дія або тест над виявленим елементом.

**Локатор** — це шлях до шуканого елементу у якій-небудь структурі. Ми з вами щодня працюємо з локаторами, приклад – файлова система на наших комп’ютерах.

Існує три типи шляхів до файлу:

* **Абсолютний шлях до файлу** — це шлях до файлу починаючи з диска із вказанням усіх папок, які проходяться. Основна перевага – включає в себе всю необхідну інформацію для пошуку необхідного файлу, починаючи від батьківського елемента (диска) і пройшовши всі вкладення до шуканого файлу. Мінус даного шляху в тому, що якщо хтось перенесе папку, то шлях стане недійсним. Тобто, цей шлях включає безліч значень, зміна яких може вплинути на сам шлях.

*C:\Users\t.radulenko\Pictures\test\_file.txt*

* **Відносний шлях до файлу** допоможе уникнути проблем попереднього шляху. Нам просто необхідно знати, де знаходиться папка Picture, і саме там ми зможемо знайти файл, який ми шукаємо. Переваги даного шляху в тому, що цей шлях більш захищений від перенесення або перейменування папок. Оскільки якщо ми знаємо, яким чином знайти необхідну папку, то ми зможемо знайти і наш файл.

*Picture\test\_file.txt*

* **Точний або короткий шлях** дозволяє знайти файл просто з пошуку, якщо ми знаємо його унікальний ідентифікатор (наприклад Ім’я). Це безперечно найкращий спосіб. Однак необхідно стежити за відсутністю дуплікацій, наприклад імені.

*test\_file.txt*

**X-Path локатори. Теорія**

**https://qalight.ua/baza-znaniy/x-path-lokatori-teoriya/**

**https://youtu.be/frVX-xgC0DU**

Приклад локатора —> *.//button[text()=’LogIn’]*

Локатор складається з наступних **елементів**:

* *TagName: button, input, h1*;
* */*— роздільник;
* *//*— запит: обери мені усі елементи;
* *\**— запит: обери мені будь-який елемент;
* *p*— параграф, елемент до якиго вписано текст;
* *a*— посилання на сторінці;
* *h1*— заголовки h1 (аналогічно з h2, h3 і т.д.);
* *.*— означає початок локатора;
* *..*— означає звернення до батьківського елементу, коли ми хочемо вийти на рівень вище, наприклад, до батьківської папки.

**Атрибути**всередині тегу, нашого елементу:

* *@id*;
* *@name*;
* *@class*и др.

**Фільтри**нашого елементу:

* *//a[text()=’some value’]*дорівнюється до запиту: знайди нам усі посилання (a) з текстом ’some value’;
* *//\*[a and b]*дорівнюється до запиту: знайди мені усі елементи, які містять елемент a і b. Також підтримується оператор *OR*.

**Функції**:

* *count()*— обчислює кількість елементів за даним локатором;
* *text()*— проводить фільтрацію за точним входженням тексту, включаючи пробіли;
* *contains()*— аналогічно фільтрує елементи, але вже за неточним входженням (у SQL = like());
* *position()*— вказує позицію елемента у структурі, проте цю функцію небажано використовувати, тому що позиція елемента може змінюватися.

**Як написати локатор і виконати пошук за локатором**

**https://qalight.ua/baza-znaniy/yak-napisati-x-path-lokator/**

**https://qalight.ua/baza-znaniy/vikoristannya-tagname/**

**https://qalight.ua/baza-znaniy/vkladenist-batkivskogo-elementu/**

**Як обрати інструмент автоматизації?**

Вибір відповідного інструменту є однією з найбільших завдань, яке слід вирішити, перш ніж перейти до автоматизації. *По-перше, визначте вимоги, вивчіть різні інструменти та їх можливості, встановіть очікування від інструменту і отримаєте підтвердження концепції.*

Наступні критерії допоможуть обрати найкращий інструмент:

* підтримка середовища;
* простота використання;
* тестування бази даних;
* ідентифікація об’єкта;
* тестування зображення;
* тестування відновлення після помилок;
* відображення об’єктів;
* використовується мова сценаріїв;
* підтримка різних типів тестів, включаючи функціональні, управління тестуванням, мобільне тестування тощо;
* підтримка кількох середовищ тестування;
* легкість налагодження скриптів програмного забезпечення для автоматизації;
* здатність розпізнавати об’єкти у будь-якому середовищі;
* великі test reports та звіти;
* мінімізована вартість навчання для обраних інструментів.

**Переваги**автоматизації тестування:

* на 70% швидше, ніж ручне тестування;
* ширше охоплення функцій програми тестуванням;
* надійність результатів;
* забезпечує акуратність, точність та логічність процедур;
* заощаджує час і кошти;
* під час виконання не потрібне втручання людини;
* збільшує ефективність;
* повторно використовувані тестові сценарії;
* можливість проведення частого та детального тестування;
* можна досягнути більшої кількості циклів розробки за рахунок автоматизації тестування;
* ранній час виходу на ринок.

Правильний вибір інструменту автоматизації, процесу тестування та команди – важливі складові для проведення успішного автоматизованого тестування. Ручні й автоматизовані методи доповнюють один одного для успішного проведення тестування.

**Література**

**1**. Автоматизоване тестування. База знань. QaLight: Центр підготовки IT фахівців: вебсайт. URL: https://qalight.ua/baza-znaniy/avtomatizovane- testuvannya/.

2. IEEE. Guide to the Software Engineering Body of Knowledge. SWEBOK, 2004. 335 p.

3. Ручне та автоматизоване тестування. База знань. QaLight: Центр підготовки IT фахівців: вебсайт. URL: https://qalight.ua/baza-znaniy/ruchne-ta-avtomatizovane-testuvannya.

4. International Software Testing Qualifications Board. ISTQB Certification –Foundation Level syllabus, ISTQB. 2023. 74 p.

5. Ляшенко Дарʼя. Автоматизоване тестування UI: стисло про головне. 2021.

Codeguida: вебсайт. URL: https://codeguida.com/post/2881.

6. Тестування UI (інтерфейсу користувача). Блог. 2021. Wezom: вебсайт. URL:https://wezom.com.ua/ua/blog/testing-ui-user-interface.

7. WebDriver. Documentation. Selenium: вебсайт. URL: https://www.selenium.dev/documentation/webdriver/.

8. Що таке Selenium WebDriver? QualityAssuranceGroup: вебсайт. URL:https://qagroup.com.ua/publications/shcho-take-selenium-webdriver/.

9. Browser Options. Documentation. Selenium: вебсайт. URL: https://www.selenium.dev/documentation/webdriver/drivers/options/.

10. Browser Navigation. Documentation. Selenium: вебсайт. URL:https://www.selenium.dev/documentation/webdriver/interactions/navigation/.

11. Working with windows and tabs. Documentation. Selenium: вебсайт. URL:https://www.selenium.dev/documentation/webdriver/interactions/windows/.

12. Browser Interactions. Documentation. Selenium: вебсайт. URL:https://www.selenium.dev/documentation/webdriver/interactions/.

13. Locator strategies. Documentation. Selenium: вебсайт. URL:https://www.selenium.dev/documentation/webdriver/elements/locators/.

14. XPath vs CSS Selector. QualityAssuranceGroup: вебсайт. URL: https://qagroup.com.ua/publications/xpath-vs-css-selector/.

**Питання для самоконтролю:**