МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних технологій

Кафедра комп'ютерних наук

Дипломна магістерська робота

на тему: Створення фреймворку для автоматизованого тестування веб-застосунку з використанням BDD-технологій

Виконав: студент групи МгІТ 2-20

спеціальності 122 Комп'ютерні науки

освітньої програми Комп`ютерні науки

Ярослав НІКІТЧЕНКО

Керівник:

к.т.н., доц. Вікторія РЕЗАНОВА

Рецензент:

д-р ф.-м.н., проф. Сергій КРАСНИТСЬКИЙ

Київ 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет мехатроніки та комп'ютерних наук

Кафедра комп'ютерних наук

Спеціальність 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри комп'ютерних наук

_____ Володимир Щербань

"_____" _____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Нікітченку Ярославу Юрійовичу

1. Тема роботи: Створення фреймворку для автоматизованого тестування веб-застосунку з використанням BDD-технологій

Науковий керівник роботи д-т, к-т техн. наук Резанова В.Г.
затверджені наказом вищого навчального закладу від "04" жовтня 2021 року № 286

2. Строк подання студентом роботи: 15.12.21

3. Вихідні дані до роботи: Розробка кафедри комп'ютерних наук. Поведінково-орієнтоване тестування (behavior-driventesting, BDD) — поширений підхід до тестування інтерактивних веб-застосунків

4. Зміст дипломної роботи: Вступ; РОЗДІЛ 1 Математичне забезпечення; РОЗДІЛ 2 Алгоритмічне забезпечення; РОЗДІЛ 3 Програмне забезпечення; Висновки.

5. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	Вікторія РЕЗАНОВА, к.т.н., доцент		
Розділ 1	Вікторія РЕЗАНОВА, к.т.н., доцент		
Розділ 2	Вікторія РЕЗАНОВА, к.т.н., доцент		
Розділ 3	Вікторія РЕЗАНОВА, к.т.н., доцент		
Висновки	Вікторія РЕЗАНОВА, к.т.н., доцент		

6. Дата видачі завдання 10.2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання	
			Студ.	*Керівн.*
1	Вступ	15.10.2021		
2	Розділ 1 Математичне забезпечення	25.10.2021		
3	Розділ 2 Алгоритмічне забезпечення	30.10.2021		
4	Розділ 3 Програмне забезпечення	10.11.2021		
5	Висновки	10.11.2021		
6	Оформлення дипломної магістерської роботи (чистовий варіант)	15.11.2021		
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	20.11.2021 до 10.12.21		
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	7.12.21		
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	30.11.2021 до 17.12.21		

Студент _____ Ярослав НІКІТЧЕНКО

Науковий керівник роботи _____ Вікторія РЕЗАНОВА

Директор НМЦУПФ _____ Олена ГРИГОРЕВСЬКА

Зміст

Анотація

Нікітченко Я.Ю. Створення фреймворку для автоматизованого тестування веб-застосунку з використанням BDD-технологій– Рукопис.

Дипломна магістерська робота за спеціальністю 122 - «Комп'ютерні науки». – Київський національний університет технологій та дизайну, Київ, 2021 рік.

Мета. Розробка програмного забезпечення для автоматизованого тестування веб-застосунку з використанням BDD-технологій.

Задачі дослідження. Дослідити особливості методології BDD. Врахувати особливості тестування із використанням BDD-технологій та без них. Порівняти їх і зробити відповідні висновки.

Практичні результати. Розроблено програмне забезпечення для автоматизованого тестування веб-застосунку з використанням BDD-технологій. В якості застосування розглянуто конкретне тестування веб застосунку, в даному випадку сайт новин BBC.

Ключові слова: програмне забезпечення, автоматизоване тестування, веб-застосунок, BDD.

ANNOTATION

Nikitchenko Y.Y. Creating a framework for automated testing of web applications using BDD technologies. It is Manuscript.

Diploma master's degree work on speciality 122 - «Computer science», Kyiv national university of technologies and design, Kyiv, 2021 year.

Purpose. Development of software for automated testing of web applications using BDD technology.

Research objectives. Investigate the features of BDD methodology. Consider the features of testing with and without BDD technologies. Compare them and make appropriate conclusions.

Results. Software for automated testing of web applications using BDD technology has been developed. BBC website was used as an application in order to show framework at work.

Keywords: software, automated testing, web application, BDD.

Вступ

Актуальність теми. Сучасний процес розробки програмного забезпечення неможливо представити без тестування. Тому пошук якнайкращого підходу, з метою спростити чи організувати цей процес, є повсякденною задачею кожного тестувальника.

Мета дослідження. Дослідження методології BDD та розробка програмного забезпечення для автоматизованого тестування веб-застосунку з використанням BDD-технологій.

Завдання дослідження. Розробити програмне забезпечення для автоматизованого тестування веб-застосунку з використанням BDD-технологій.

Об’єкт дослідження. Програмний продукт для автоматизованого тестування веб-застосунку з використанням BDD-технологій.

Предмет дослідження. Реалізація програмного забезпечення.

Методи дослідження. Дослідження методів для створення для автоматизованого тестування веб-застосунку з використанням BDD-технологій. Програмне забезпечення розроблялося мовою Java у середовищі InteliJIdea.

Практична цінність. Прискорення та спрощення процесу тестування, з метою зменшити кількість витрат(часових та грошових) під час розробки ПЗ.

Елементи наукової новизни. Реалізація програмного для автоматизованого тестування веб-застосунку з метою подальшого розширення та оновлення новими тестами, щоб протестувати веб-застосунок якнайкраще, щоб зменшити кількість витрат на виправлення багів після релізу продукту.

Практична значущість роботи. Створене програмне забезпечення дозволяє значно прискорити та спростити процес тестування веб-застосунку, що може використовуватися під час процесу створення та тестування нового програмного продукту програмістами та тестувальниками.

Апробація результатів роботи. Розроблено програмне забезпечення для для автоматизованого тестування веб-застосунку з використанням BDD-технологій. В якості застосування розглянуто конкретне тестування веб застосунку, в даному випадку сайт новин BBC.

Тестування — це широкий процес, який складається з декількох взаємопов’язаних процесів. Іншими словами сукупності процесів.

Часто говорячи про тестування, люди уявляють собі картинку, в якій спеціаліст перевіряє чи за всіма параметрами програма працює ідеально. Але якість не є абсолютною, це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід виконати перезапуск тестів, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціальної додаткової перевірки. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений професором Едсжером Дейкстри у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!".

Отож мета і завдання моєї роботи - це розглянути різноманітні підходи до тестування програмного забезпечення, провести детальний опис кожного та порівняти їх між собою, а також розглянути особливості BDD, їх використання та різновиди.

Ця робота складається з 4 розділів, в кожному з яких будуть описані основні підходи до тестування програмного забезпечення, які, зрештою, утворять загальну картину про те, як відбувається перевірка програми на працездатність та відповідність заявленим вимогам.

В першому розділі розглядається загальні відомості щодо тестування програмного забезпечення. В ньому детально розписано, що таке загалом тестування, навіщо воно існує та в якому вигляді його використовують в сучасності.

В другому розділі розповідається про основні принципи і засади BDD. А саме, описується цей процес розробки програмного забезпечення загалом, а також наводяться приклади використання у вигляді фреймворків.

В третьому розділі відбувається детальний огляд кожного з BDD-фреймворків, їх особливостей, а також проводиться їх порівняльна характеристика.

В четвертому розділі описується приклад реалізації тестування веб-застосунку з використанням BDD-фреймворку.

РОЗДІЛ 1 Математичне забезпечення. Теоретичні відомості щодо тестування програмного забезпечення

1.1 Основні поняття та визначення

Тестування програмного забезпечення є невід'ємною частиною створення програмного продукту. Від того, наскільки досконало створені тести, залежить те, як скоро проект буде зданий остаточно, і чи буде необхідність згодом усувати помилки.

Тестування програмного забезпечення - це процес, що використовується для виміру якості розроблюваного програмного забезпечення. Зазвичай, поняття якості обмежується такими поняттями, як коректність, повнота, безпечність, але може містити більше технічних вимог. Тестування - це процес технічного дослідження, який виконується на вимогу замовників, і призначений для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок.

Тестування так само можна описати як процес верифікації та валідації того чи іншого програмного продукту, щоб дізнатися на скільки точно він задовольняє всім встановленим вимогам.

Верифікація (Verification – узгодження) – це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу (чи виконуються наші цілі, терміни, завдання, по розробці проекту, визначені на початку поточної фази.)

Валідація (Validation – затвердження) – це визначення відповідності ПЗ очікуванням і потребам користувача, вимогам до системи.

Тестоване програмне забезпечення повинно проходити кожен з етапів тестування, обумовлених продуктовнерами, менеджментом компанії розробника та тест-дизайнерами для того, щоб вважати програмний продукт відносно якісним або придатним до використання.

Головними типами методів тестування програм вважаються: тестування “білої скриньки” та “чорної скриньки”.

У випадку “білої скриньки” об'єктом тестування тут є не зовнішня, а внутрішня поведінка програми. Перевіряється коректність побудови всіх елементів програми та правильність їхньої взаємодії один з одним. Зазвичай аналізуються керуючі зв'язки елементів, рідше — інформаційні зв'язки. Тестування за принципом “білого ящика” характеризується ступенем, в якому тести виконують або покривають логіку (вихідний текст) програми.

Особливостями тестування “білої скриньки” є те, що воно засноване на аналізі керуючої структури програми. Програма вважається повністю перевіреною, якщо проведено вичерпне тестування маршрутів (шляхів) її графи управління.

Основним же місцем програми тестів “чорної скриньки” — інтерфейс програмного забезпечення.

Ці тести демонструють:

- Як виконуються функції програми.
- Як приймаються вихідні дані.
- Як виробляються результати.
- Як зберігається цілісність зовнішньої інформації.

Тестування «чорної скриньки» дозволяє отримати комбінації вхідних даних, які забезпечують повну перевірку всіх функціональних вимог до програми.

Окрім того, програмне забезпечення може бути протестоване в цілому, в компонентах, одиницях або в живій системі.

Але узагалі на сьогодні розрізняють близько 150 видів тестування програмного забезпечення, щоб за необхідності протестувати програмний продукт від А до Я на усі 100%

1.2 Роль тестування в розробці програмного забезпечення

Під час користування будь-якою програмою ми навіть не замислюємося про те, чого та чи інша програма працює без помилок, зависань тощо. Саме в цьому аспекті тестування відіграє далеко не останню скрипку.

Чому ж тестування в рамках даного процесу настільки важливе? Ось кілька причин:

1) Тестування дозволяє перевірити, чи правильно реалізовано усі вимоги до програмного забезпечення, що розроблялось.
2) Тестування допомагає у виявленні помилок та забезпечує їх розпізнавання і вирішення до етапу розгортання програмного забезпечення.
3) Тестування пом’якшує наслідки та ризики втрат, якщо програмний продукт все ж випустили по неправильним вимогам. В такому випадку намагаються частково виправити, переоцінити та покращити програму.
4) Тестування допомагає перевірити належну інтеграцію та взаємодію програми з навколишнім середовищем.

Звідси основна мета тестування програмного забезпечення, яка полягає у намаганні виміряти рівень якості програмного продукту з точки зору основних вимог.

Люди схильні помилятися, людські помилки можуть призводити до порушення нормальної роботи програмного забезпечення на всіх стадіях розробки, причому наслідки цього можуть бути найрізноманітнішими — від незначних до катастрофічних.

Для програмного забезпечення будь-якої складності неможливо вилучити всі проблеми. Проте тестування допомагає виявити більшість помилок, з якими може зіткнутися користувач, й потенційно зменшує ризик виникнення проблем з програмою у майбутньому.

1.3 Види тестування

У залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:

- Функціональні.
- Нефункціональні.
- Пов'язані зі змінами.

Функціональні тести базуються на функціях та особливостях, а також на взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному, інтеграційному, системному і приймальному. Такі види перевірки розглядають зовнішню поведінку системи. Одні з найпоширеніших видів функціональних тестів:

- Тестування безпеки - стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту програми, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків.
- Тестування взаємодії - це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності та інтеграційне тестування.

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. У цілому, це перевірка того, як система працює. Далі перераховані основні види нефункціональних тестів:

- Тестування продуктивності, яке включає в себе тестування навантаження, стресове тестування, тестування стабільності або надійності та об'ємне тестування.

- Тестування установки, яке спрямоване на перевірку успішної інсталяції та настройки, а також оновлення або видалення програмного забезпечення.
- Тестування зручності користування - це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблюваного продукту в контексті заданих умов.
- Тестування на відмову і відновлення, яке перевіряє тестований продукт з точки зору здатності протистояти й успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку.
- Конфігураційне тестування - ще один вид традиційного тестування продуктивності. У цьому випадку замість того, щоб тестувати продуктивність системи з точки зору навантаження, тестується ефект впливу на продуктивність змін у конфігурації.

Після проведення необхідних змін, таких як виправлення дефекту, програма повинна бути протестована для підтвердження того факту, що проблему було дійсно вирішено. Нижче перераховані види перевірок, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності або правильності здійсненого виправлення помилки:

- Димове тестування. Поняття цього види перевірки пішло з інженерного середовища. При введенні в експлуатацію нового "заліза" вважалося, що тестування пройшло вдало, якщо з установки не пішов дим. В області ж програмування, воно спрямоване на поверхневу перевірку всіх модулів на предмет працездатності та наявність швидко встановлюваних критичних і блокуючих дефектів.
- Регресійне тестування - вид тестування спрямований на перевірку змін, зроблених у додатку або навколишньому середовищі, для

підтвердження того факту, що існуюча раніше функціональність працює як і раніше.

- Тестування збірки, яке спрямоване на визначення відповідності випущеної версії критеріям якості для початку тестування. За своєю метою є аналогом димового тестування, спрямованого на приймання нової версії для подальшої перевірки або експлуатації.
- Санітарне тестування - вузьконаправлене тестування, яке достатнє для доказу того, що конкретна функція працює згідно заявлених у специфікації вимог.

1.4 Етапи тестування для тестувальника

Процес тестування можна поділити на 4 етапи:

- Розслідування.
- Дослідження.
- Реалізація.
- Спілкування.

Далі розглянемо кожен з них більш детально.

Процес тестування в чомусь схожий із детективним розслідуванням. Процес перевірки програмного забезпечення не завжди може бути передбачуваним, а кінцевий результат цілком відомим.

Але робота тестера, полягає у розкритті таємної інформації на усьому шляху, яка допоможе людям приймати правильні рішення. Це набагато більше, ніж лише опрацювати специфікацію і шляхом порівняння зробити свою оцінку.

Тестувальнику під час роботи потрібно мислити критично, задавати складні питання, зважувати ризики, помічати такі речі, які на перший погляд здаються несуттєвими, однак котрі при суворій експертизі набагато важливіші, ніж здавалося спочатку й потребують подальшого дослідження.

Тестування програм не повинно сприйматися виключно як завдання пройти тест-кейси складені по опрацюванню вимог. Наскільки б вимоги не видавалися повними, вони ніколи не стануть вичерпним переліком, який

наперед передбачить що програмне забезпечення буде робити в реальності. Завжди будуть нюанси, які не зазначені, які не передбачили або опустили. Звідси тест кейси не можуть повністю покрити програму.

Під час перевірки та тестування системи повинні поєднуватися з розслідуванням та дослідженням. Дослідницька перевірка передбачає одночасне вивчення, тест-дизайн та виконання тестів. Спеціаліст аналізує додаток, знаходить нову інформацію, вивчає систему й у цьому процесі знаходить нові речі, які б варто було перевірити.

Тестування є цінним на всіх етапах життєвого циклу розробки програмного забезпечення, а не тільки на етапі кодування. І більша частка цінності зосереджується на особі, яка його проводить — інтелекті спеціаліста, який допомагає виявляти проблеми вчасно, тобто якомога раніше. Автоматизоване тестування та комп'ютер ще не скоро на цій ланці замінить людину.

Робота спеціаліста у цій галузі — це постійне випробування, бо інколи потрібно придумувати вигадливі трюки, щоб щось протестувати або зрозуміти, як воно повинне працювати. Крім того, необхідно мати уміння експериментувати та знаходити найкращі інструменти для роботи.

Через таку "творчість" роботу тестувальника складно оцінити. Те ж саме стосується оцінки якості програмного забезпечення, яку робити за допомогою показників — оманливо.

Левова частина роботи тестера — це спілкування. Вони спілкуються, надають інформацію про якість програмного продукту людям різних ролей, з різним рівнем підпорядкування по вертикалі управління, з різним рівнем знань. І робиться все це для того, щобвони на основі інформації, котру їм надали тестери змогли прийняти правильні рішення.

Таким чином, кожен тестувальник повинен однаково удосконалювати як свої технічні навички, так і вміння спілкуватися з іншими людьми. Він зобов'язаний: чітко висловлюватися, обґрунтовано висловлювати

припущення на основі фактів, своєю поведінкою уникати непорозумінь,бути неконфліктним таза неправильні дії без проблем вибачатися.

Висновки до розділу 1

Підсумовуючи все вище сказане у “Розділі 1” можна зробити наступні висновки що:

- тестування програмного забезпечення є невід'ємною частиною створення програмного продукту;
- мета тестування програмного забезпечення полягає у намаганні виміряти рівень якості програмного продукту з точки зору основних вимог;
- тестування є цінним на всіх етапах життєвого циклу розробки програмного забезпечення, а не тільки на етапі кодування;
- у залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:
    1) функціональні;
    2) нефункціональні;
    3) пов'язані зі змінами;
- процес тестування можна поділити на 4 етапи:
    1) Розслідування.
    2) Дослідження.
    3) Реалізація.
    4) Спілкування.

РОЗДІЛ 2 Алгоритмічне забезпечення. Принципи і засади BDD, порівняння існуючих BDD-фреймворків

2.1 Основні поняття та визначення

Сучасні проекти все частіше висувають високі вимоги до покриття автоматичними тестами. У наш час писати тести не просто ознака хорошого тону, але одна з вимог, яка пред'являється до коду. Все частіше ми чуємо такі слова, як TDD (TestDrivenDevelopment) і BDD (BehaviourDrivenDevelopment) і багато строго слідують цим підходам в розробці.

BDD є розширенням Test-DrivenDevelopment (TDD), який підкреслює розробку функцій на основі історії користувача та написання коду, що забезпечує вирішення реальних проблем.

Це означає, що клієнт або менеджер фірми повідомляє про своє бачення, і тоді розробник повинен визначити модель поведінки для досягнення визначених бізнес-цілей. Потім тестер залучається до роботи, щоб дізнатися, чи відповідає задана модель утвореному коду.

BDD - behaviour-drivendevelopment - це розробка, заснована на описі поведінки. Тобто, є спеціальна людина (або люди) який пише опис виду: "Я як користувач хочу, щоб при натисненні кнопки пуск показувалося меню".

Наприклад, якщо клієнт сказав, що знайшов у своїх дослідженнях, що у них є багато користувачів похилого віку, які використовують їх програму, і їм потрібен спосіб зробити її доступнішою, то ваш крок як розробника, що слідує за моделлю BDD, буде розглянути, яких змін набуде поведінка, якщо програма набуде більших шрифтів та елементів, які можна легше натискати.

BDD приділяє велику увагу співробітництву команди. Переконатися, що історії користувачів та поведінкова модель повідомляються з ділової сторони на технічну, є невід'ємною частиною успішного BDD.

Основною ідеєю даної методології є поєднання в процесі розробки чисто технічних інтересів і інтересів бізнесу, дозволяючи тим самим керуючому персоналу і програмістам говорити на одній мові.

Для спілкування між цими групами персоналу використовується предметно-орієнтована мова, основу якої становлять конструкції з природної мови, зрозумілі неспеціалісту, яка зазвичай виражає поведінку програмного продукту і очікувані результати.

Подібно до TDD, BDD виступає за те, що тести повинні бути написані першими, що добре для високого випробувального покриття. У BehaviourDrivenDevelopment ви створюєте накопичувальні приймальні випробування, що означає, що найчастіше команди, які слідують за ними, використовують автоматизацію тестування.

Хоча й виникають помилкові уявлення про автоматизації тестування, BDD додатково доводить важливість тестерів. Тестери не тільки повинні переконатися, що код працює, але вони також повинні розглянути проблеми, які він вирішує.

Це означає, що BDD тестування перевіряє не тільки виконуваність функцій, а й також початкову поведінку, для якої вони будується. Замість того, щоб просто думати про те, чи працює певний метод, спеціалісти повинні думати про це в контексті сценарію, в якому він використовується.

2.2 Переваги BDD

Звичайно BDD – це дуже зручна технологія, яка полегшує життя замовнику та виконавцю, але все на світі має свої позитивні та негативні сторони. Спочатку розглянемо переваги.

По-перше, BBD зосереджує увагу на кінцевому користувачеві та його потребах. Інженерам легко заплутатися в деталях реалізації та вибору архітектури. Вони можуть втратити з уваги той факт, що всі ці речі в кінцевому рахунку повинні обслуговувати користувача програмного забезпечення. BDD змушує вас зробити крок назад і подивитися на програму з точки зору замовника.

Ще одна цікава річ про BBD полягає в тому, що документація вбудована. Пишучи список специфікацій високого рівня, ви надаєте опис того, що ваша програма насправді робить простими словами, які розуміє кожен член вашої

команди. Ви можете використовувати ці специфікації для створення живої документації, яку можна переглянути всіма, хто працює над проектом.

Крім того, як BDD, так і тестова розробка (TDD) дозволяють без особливих втрат змінювати код. Наявність комплексного покриття дозволяє дуже ефективно редагувати речі, не турбуючись про те, що відбувається. Ваші тести покажуть вам, що відбувається відразу. Якщо Ви не впевнені, що Вам потрібен модуль, можна його вилучити і відразу побачити, що відбувається. Тести дадуть відгуки, система повідомить, що вийшло, і можна виправити ці речі і залишити все інше.

Тобто усі вище перераховані переваги можна виокремити у наступні пункти:

1) Ви більше не визначаєте "тест", а визначаєте "поведінку".
2) Краще спілкування між розробниками, тестерами та власниками продуктів.
3) Оскільки BDD пояснюється за допомогою простої мови, крива навчання буде набагато коротшою.
4) Будучи нетехнічним за своєю природою, BDD може охопити широку аудиторію.

2.3 Недоліки BDD

Тепер розглянемо недоліки, яких теж не мало.

Іноді буває дуже важко дійти згоди усією командою. Кожен починає помічати проблеми з новою технологією, яких не було в старій, до якої звикли і користуються щодня. Переваги чогось нового відходять на другий план. І це дуже типово, коли ви здійснюєте будь-які зміни в межах компанії, організації тощо.

Пуризм також є проблемою. Ви отримуєте ситуації, коли ортодоксальний підхід не обов'язково є найкращим. Люди не чудово визнають винятки з правил: вони або занадто жорсткі, взагалі не допускають жодних виключень, або дозволять все і тоді робота виходить з під контролю.

Також можна розглянути приклад недоліку на випадку з BDD-фреймворком Cucumber, який наразі є одним із найпопулярніших. Багато людей, які його використовують неправильно, просто відмовляються від нього відразу, що змушує їх не любити цей інструмент. Історично цього достатньо, щоб люди більше не поверталися до нього. Певною мірою це стосується BDD взагалі. Загалом є багато поганих практик і поганих реалізацій.

Отже, підсумувати недоліки можна наступними пунктами:

1) Для роботи в BDD необхідний попередній досвід TDD.
2) Якщо вимоги не визначені належним чином, BDD може бути неефективним.
3) Тестери, що використовують BDD, повинні мати достатні технічні навички.

2.4 BDD-scenarios (сценарії)

Ваше перше питання, ймовірно: "Що таке сценарії BDD?". Це справедливе питання.

Сценарій BDD - це письмовий опис поведінки вашого продукту з одного або декількох позицій користувачів. Вони призначені для зниження вартості перекладу та полегшення вашим інженерам розуміння вимог для забезпечення якості належної перевірки.

Сценарії - це спосіб пояснити, як дана функція повинна поводитися в різних ситуаціях або з різними вхідними параметрами. Вони структуровані навколо моделі контексту-наслідку і написані в спеціальному форматі Gherkin.

Використання сценаріїв BDD означає, що вимоги та тести можуть бути об'єднані в 1 специфікацію. У деяких випадках письмові сценарії можуть бути легко перетворені в автоматизовані тести. Сценарії BDD мають тенденцію до певного формату. Формат досить прямолінійний, і з невеликою практикою ви зможете написати свої власні.

Feature files - це текстові файли з розширенням .feature, які можна відкрити будь-яким текстовим редактором, а також читати будь-яким інструментом BDD, наприклад, Cucumber, JBehave або Behat.

Feature files повинні починатися з контексту функції (що є по суті історією), за якою слід писати сценарій у наступному форматі:

Feature: Some terse yet descriptive text of what is desired

In order to realize a named business value

As an explicit system actor

I want to gain some beneficial outcome which furthers the goal

Scenario: Some determinable business situation

Given some precondition

And some other precondition

When some action by the actor

And some other action

And yet another action

Then some testable outcome is achieved

And something else we can check happens too

Метою feature files є документування обговорюваних сценаріїв для того, щоб дати вказівку про те, скільки роботи необхідно для успішного виконання певної функції. Feature files також є драйверами для автоматизованих тестів. Feature files також слугують визначенням зробленого (DoD), тобто, коли всі сценарії були успішно впроваджені та перевірені, ми можемо позначити історію як виконану.

Також особливістю файлів сценарію є те, що не має значення, хто насправді пише їх, бо це може бути будь-який член команди замовника. Однак, зміст, є невід'ємною частиною функції файлів, тому вони мають обговорюватись за наявності двох сторін. Отримання загального розуміння того, що має бути зроблено є ключовим елементом.

Як правило, існують два способи написання feature files – імперативний і декларативний

Імперативний стиль написання файлу об’єкта, дуже детальний, містить деталі низького рівня і занадто багато інформації.

Плюси: людина, що читає файл функції, може стежити за кроком

Мінуси: Через занадто багато деталей читач може втратити точку історії сценарію та тестів. Файл об’єкта стає занадто великим, його важко підтримувати.

Декларативний стиль написання файлу об’єкта є лаконічним і містить у собі лише актуальну інформацію про історію.

Плюси: декларативний стиль більш читабельний, оскільки містить менше кроків у сценарії. Читач може легко зрозуміти масштаби тесту і швидко визначити, чи відсутні будь-які ключові елементи.

2.5 Різновиди BDD-фреймворків

Програмний фреймворк - це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми. Відповідно — програмний фреймворк може містити в собі також допоміжні програми, деякі бібліотеки коду, скрипти та загалом все, що полегшує створення та поєднання різних компонентів великого програмного забезпечення.

BDD теж не вийняток і має дуже багато різних фреймворків.

Cucumber - це behavior driven development(BDD), з відкритим вихідним кодом який працює з Ruby, Java, .NET, Flex або веб-додатками, написаними будь-якою мовою.

givwenzen-flex - це система тестування прийому BDD стилів для Flex.

GivWenZen дозволяє користувачеві користуватися BDD Given When Then словником та "простим текстом", щоб допомогти команді отримати правильні слова і створити універсальну мову для опису та тестування бізнес-домену.

Instinct - це Behaviour Driven Development (BDD) структура для Java. Натхненна RSpec, Instinct забезпечує гнучку анотацію контекстів, специфікацій та акторів; має підтримку плагіна IntelliJ IDEA.

gospecify надає синтаксис BDD для тестування вашого коду Go.

Spectacular це Behavior Driven Development (BDD) утиліта, яка об'єднує декілька різних типів тестових фреймворків в 1.

JSNSpec - це JavaScript фреймворк на платформі .Net Specification, що підтримує розробку з BDD.

PHPSpec - це BDD-фреймворк для PHP. PHPSpec дозволяє писати приклади виконуваних файлів, що відображають специфікації бажаної поведінки описаного вихідного коду.

dSpec - це BDD-фреймворк для Delphi / Pascal, побудована як розширення DUnit.

Specter - це фреймворк специфікації об'єктної поведінки. Вона вмикає BDD, дозволяючи розробникам писати виконувані специфікації для своїх об'єктів, перш ніж фактично їх реалізувати.

StoryQ - це фреймворк C # 3.5 для керування BDD. Подібний до Behave, але загалом простіший. Історії можна створювати або у вигляді звичайного тексту, а потім пасивно генерувати як C # код або безпосередньо вводити в тестах. Вихідні дані - це консолі або html файли.

NBehave - це еволюція тестованої розробки (TDD) і проектування, що керується прийняттям тестів, і має на меті зробити ці практики більш доступними та інтуїтивними для новачків і експертів.

JBehave - це фреймворк на основі Java, розроблений для заохочення співпраці між розробниками, бізнес-партнерами та іншими членами команди через автоматизовані сценарії.

JNarrate призначений для написання специфікацій поведінки в Java.

Narrative є основою для побудови тестів з BDD у вільній Java.

specs надає BDD-фреймворк для мови Scala.

Steak - це смачне поєднання RSpec і Capybara для Acceptance BDD

UISpec є основою BDD для iPhone, що забезпечує повне автоматизоване тестування, яке керує реальним інтерфейсом iPhone. Він моделюється за дуже популярним RSpec для Ruby.

NSpecify - це BDD-фреймворк, розроблений в C # .NET.

JSSpec – це JavaScript BDD-фрецмворк.

Easyb - це BDD-фреймворк для платформи Java. easyb прагне включити виконувану, але читабельну документацію.

RSpec є BDD-фреймворком для Ruby.

2.6 Детальний опис найпопулярніших фреймворків

TestLeft:

- TestLeft - інструмент для тестування автоматизації користувацького інтерфейсу в IDE.
- Забезпечує створення моделей додатків для веб-і настільних додатків у два кліки.
- Легко інтегрується з Cucumber, SpecFlow та JBehave.
- Має підтримку .NET, C #, Java, Jenkins та інших.
- Безпроблемно вбудовується в будь-яке середовище розробки екосистеми DevOps.

Створює стислий і коротший код для надійної системи тестування.

Cucumber:

- Вільний для використання.
- Cucumber допомагає в написанні тестів, які легко зрозуміти будь-кому, незалежно від технічних знань, якими вони володіють.
- Зацікавлені сторони, власники бізнесу, тестери та розробники мають змогу працювати над постановкою проблеми, щоб отримати кращий набір поведінки. Після цього набори поведінки змінюються на умови прийняття тесту на Cucumber.
- Cucumber зазвичай використовує Ruby як мову програмування. Однак також можна використовувати інші мови, такі як JAVA, Groovy тощо.
- Команда тестування та команда розробників беруть участь у написанні та розробці умов тестування.
- Подібно до Selena, Cucumber обмежений лише для веб-автоматизації.

EasyB:

- EasyB - це структура, яка використовує історії як одиниці верифікації.
- EasyB в основному написано за допомогою Groovy і сумісний для роботи з мовами Java або Groovy.
- EasyB можна використовувати для створення історій користувача, оголошення специфікацій тощо.
- EasyB використовує сценарій Groovy, специфікації можуть бути написані більше англійською мовою, ніж у програміст-орієнтованих фреймворках. Це робить EasyB одним із найпопулярнішим в BDD.
- Після того, як специфікація введена, вона називається специфікацією очікування. Вони додаються до системи після реалізації.
- Для EasyB не існує інтеграції IDE.
- EasyB також надає інструмент звітності, яким дуже зручно отримувати дані у вигляді XML і HTML.

JDave:

- JDave працює з JUnit, а це означає, що він може працювати легко в Eclipse.
- JDave інтегрується з JMOCK2 і Hamcrest як макетна рамка і відповідна бібліотека.
- JDave є двигуном специфікації, і кожен сценарій показує поведінку класу, на відміну від рамки подій, подібних до історії, як Cucumber.
- JDave досить легкий в освоєнні, і специфікації були написані повністю по-своєму.
- Однак, коли до уваги береться BDD, JDave стає дещо більш орієнтованим на розробника фреймворком.

Concordion:

- Це один з найпотужніших інструментів для написання сценаріїв автоматизації тестування в проектах на основі JAVA.
- Concordion інтегрується з фреймворком JUnit і, отже, може використовуватися зі звичайними середовищами JAVA, такими як Eclipse, Netbeans і т.д.

- Concordion теж допомагає в написанні специфікацій. Але тут специфікації написані в HTML.
- Приймальні тести записуються в коді приладів, який є нічим іншим, як мовою JAVA.
- Concordion надає зовнішні API для розширення функціональних можливостей.
- Деякі з прикладів - підтримка Excel. Завдяки цьому, специфікації можуть бути записані в аркуші Excel і можуть бути використані звідти. Подібним чином, існують API для підтримки захоплення журналів, знімків екранів і т.д.
- Оскільки специфікації написані в HTML, специфікації документів можуть бути гіперпосиланням на HTML.
- Concordion - це чистий JAVA-фреймворк, який підтримує деякі дуже приємні для роботи варіанти звітів.

JBehave:

- Jbehave – BDD-фреймворк для JAVA. Це Open source framework, створений Dan North в 2003 році.
- Він має дві складові: Jbehave Web і Jbehave Main.
- Має чисто Java-реалізацію і має підтримку IDE.
- Jbehave має звітність, і звіти можуть генеруватися в XML, HTML або в текстовому режимі.
- Jbehave може легко інтегруватися до Selenium для запуску тестових сценаріїв на веб-додатках.
- Оскільки це каркас на основі JAVA, його можна запускати на IDE, таких як Eclipse, Netbeans і т.д.

Fitnesse:

- Fitnesse є основою автоматизації відкритих джерел на основі Framework for Integrated Test від Ward Cunningham.
- Fitnesse допомагає в автоматизації тестування прийому при інтеграції на бізнес-рівні.

- Fitnesse написано на JAVA і поєднано у файл JAR. Виконуваний файл JAR містить такі елементи, як веб-сервер, тестування, вікі-движок тощо.
- FIT і SLIM є двома тестовими системами, які входять до складу Fitnesse.
- FIT - старший брат і застарілий зараз. Немає подальшого розвитку, пов'язаного з FIT.
- SLIM, з іншого боку, це легка версія тестової системи FIT.

BeanSpec:

- BeanSpec - простий інструмент, який обробляє складні специфікації в собі.
- BeanSpec написано на Java і, отже, може бути оброблено з середовищами розробки як Eclipse і Netbeans.
- BeanSpec має власний внутрішній звіт і може бути згенерований наприкінці виконання.

2.7 Порівняння Cucumber та JBehave

Офіційна документація програмного засобу є найважливішим свідченням доступності громадськості, щоб мати уявлення про можливості та найкращі практики. Офіційні веб-сайти обох інструментів дуже багаті та організовані з інформацією.

Але в практичному середовищі загальним користувачам потрібні більш розроблені вказівки, статті в блогах і підтримка спільноти. Основною проблема, пов'язана з JBehave це форуми і блоги, написані деякий час назад і не в актуальному стані.

Почавши реалізовувати зразки проектів з обома інструментами, можна зіткнутися з багатьма практичними проблемами з JBehave і може бути важко знайти правильні шляхи вирішення. Але для Cucumber, реалізація буде прямою і менш болючою.

Gherkin - це домен-специфічна мова (DSL), яку практикуючі в BDD використовують у всьому світі як граматику, зручну для читання. Gherkin має

свій власний спосіб організувати гнучкі історії користувачів, використовуючи правила форматування, такі як функція, сценарій, кроки, приклади тощо. І JBehave і Cucumber підтримують стандарт Gherkin і його мовні правила. У JBehave також є своя граматика. Але дуже рекомендується дотримуватися синтаксису Gherkin, коли ви пишете свої історії користувача, оскільки Gherkin є прийнятим DSL в BDD.

Cucumber має вбудовану підтримку функцій Java 8 зі своїм модулем cucumber-java8. Розробники, які люблять використовувати Lambda-вирази замість звичайних методів Java, вважають цю функцію дуже привабливою в даному фреймворці. Крім того, оскільки Лямбда-вирази вбудовані в Java 8, ймовірно, її реалізація повинна бути більш ефективною, ніж визначені користувачем методи.

Більшість користувачів роблять скарги на плагіни JBehave IDE (особливо для користувачів IntelliJ IDEA). Плагіни ж Cucumber IDE працюють дуже гладко.

Звітність також відіграє важливу роль у досягненні успіху інструменту BDD. Процес формування звіту в Cucumber досить прямолінійний, і результат є інформативним і зрозумілим. Протокол події в фреймворці - це формат обміну даними між компонентами екосистеми. Програми, що підтримують протокол, можуть бути виробниками або споживачами.

З іншого боку, проблема JBehave - його звіти виглядають трохи старомодно. Безумовно, можна реалізувати плагін для покращення та налаштування звітів. Але коли Cucumber дає дуже хороші звіти відразу після встановлення, то у випадку з JBehave необхідно витратити деякий час.

Обидва інструменти будуть мати проблеми з продуктивністю, коли кількість тестових сценаріїв збільшується у вашому проекті. Переконайтеся, що сценарії та функції BDD організовані дуже акуратно. Надайте перевагу тій тестовій системі, яка зручніша вам для ефективної організації і запуску тестів. Золотого правила для вирішення таких практичних питань немає.

Тому, переконайтеся, що ви взаємодієте з фреймворками правильно та організовано.

JBehave також має багато додаткових конфігурацій для тонкої настройки інструменту BDD відповідно до ваших уподобань. Але з тими конфігураціями приходить й велика складність теж.

Підсумовуючи все вище перераховане, можна зробити висновок, що JBehave – це готовий продукт, який має свою аудиторію, але, на жаль, наразі майже не оновлюється. В той же час Cucumber – це фреймворк, який розвивається і досі, набираючи з кожним днем своїх прихильників.

Висновки до розділу 2

Підсумовуючи все вище сказане у “Розділі 2” можна зробити наступні висновки:

- програмний фреймворк - це готовий до використання комплекс програмних рішень, включаючи дизайн, логіку та базову функціональність системи або підсистеми;
- BDD методологія має дуже багато різних фреймворків, серед найпопулярніших:
  1) Cucumber
  2) JBehave
  3) Spock
  4) Fitnesse
  5) JGiven
- золотого правила вибору оптимального фреймворку - немає. Тому, обирайте той, з яким ви взаємодієте найбільш правильно та організовано.
- сучасні проекти все частіше висувають високі вимоги до покриття автоматичними тестами;
- основною ідеєю методології BDD є поєднання в процесі розробки чисто технічних інтересів і інтересів бізнесу, дозволяючи тим самим керуючому персоналу і програмістам говорити на одній мові;

- переваги BDD:
    1) ви більше не визначаєте "тест", а визначаєте "поведінку";
    2) краще спілкування між розробниками, тестерами та власниками продуктів;
    3) оскільки BDD пояснюється за допомогою простої мови, крива навчання буде набагато коротшою;
    4) будучи нетехнічним за своєю природою, BDD може охопити широку аудиторію;
- недоліки BDD:
    1) для роботи в BDD необхідний попередній досвід TDD;
    2) якщо вимоги не визначені належним чином, BDD може бути неефективним;
    3) тестери, що використовують BDD, повинні мати достатні технічні навички;
- сценарії - це спосіб пояснити, як дана функція повинна поводитися в різних ситуаціях або з різними вхідними параметрами (вони структуровані навколо моделі контексту-наслідку і написані в спеціальному форматі Gherkin).

РОЗДІЛ 3 Програмне забезпечення

3.1 Опис використаного фреймворку

Для того, щоб протестувати веб-застосунок було використано BDD-фреймворк, а саме один із найпопулярніших - Cucumber.

Cucumber - це платформа тестування, яка підтримує поведінково-орієнтовану розробку (BDD). Це дозволяє нам визначати поведінку програми в простому змістовному тексті англійською мовою, використовуючи просту граматику, визначену мовою під назвою Gherkin. Сам Cucumber написаний на Ruby, але його можна використовувати для "перевірки" коду, написаного і на інших мовах, включаючи, але не обмежуючись, Java, C# та Python.

Переваги Cucumber перед іншими інструментами:

1) Cucumber виступає містком між діловою та технічною мовою. Ми можемо досягти цього, створивши тестовий кейс у вигляді звичайного англійського тексту.
2) Написати свій перший тестовий сценарій можна без знання будь-якого коду, що також дозволяє залучати людей, які не знають мов програмування.
3) На відміну від інших інструментів, Cucumber служить для наскрізного тестування.
4) Завдяки простій архітектурі тестового сценарію, даний фреймворк забезпечує повторне використання коду.
5) Різним командам у проекті необхідна спільна мова для вираження вимог. За допомогою даного BDD-фреймворку ми маємо змогу одночасно просто донести, що саме буде зроблено в рамках тестування, до бізнес-групи, і чітко пояснити задачу тестувальникам і розробникам, щоб усунути більшість неясностей;
6) За допомогою Cucumber члени команди можуть робити мозковий штурм, з метою придумати більше тестових сценаріїв. Врахувавши більше деталей, ви маєте можливість візуалізувати систему

якнайкраще, а отже, у підсумку ви створюєте більше можливих користувацьких сценаріїв.

7) Також гарно написані тести за BDD технологією можна потім використати при написанні проектної документації.

Загальнопоширеною тенденцією програмістів є розробка певних функцій та написання тестового коду пізніше. З цього можна зробити висновок, що покриття коду тестами може зайняти багато часу та бути складним. Таким чином, тестування буде відкладено до випуску. Після чого буде проведена швидка, але неефективна перевірка написаного коду розробником.

Щоб подолати цю проблему, було створено Cucumber BDD (Behavior Driven Development). Це полегшує весь процес тестування для розробника і тестувальника.

У Cucumber BDD все, що ви пишете, повинно зводитись до кроків Дано-Коли-Тоді(Given-When-Then). Кожен тест можна розділити на три секції.

Given section:

- Перш за все ми хочемо вказати контекст, в якому ми хотіли б перевірити функціональність. Тут потрібно вказати параметри вашої системи / компонента, які впливають на функціональність, що знаходиться під тестом.

When section:

- Тут ми можемо вказати, що ми хочемо перевірити. Іншими словами, яку взаємодію з досліджуваним об'єктом / компонентом ми хотіли б вивчити.

Then section:

- Тепер ми перевіряємо, чи повернуте методом правильне або тест провалився.

Основні файли в Cucumber - це текстові файли з розширенням .feature, які містять опис сценаріїв. Давайте розглянемо тестовий сценарій написаний за допомогою Gherkin на прикладі калькулятора:

Feature: Calculator

Scenario Outline: Sum of the two numbers

Given two numbers <a> and <b>

When we try to find sum of our numbers

Then result should be <result>

Examples: | a | b | result |

| 3 | 2 | 5 |

Реалізація ж кроків описаних у .feature відбувається у відповідному .java файлі:

```
public class CalculatorSteps {
    private Calculator calc;
    double a;
    double b;
    double result;
    @Given("^two numbers (\\d) and (\\d)")
    public void given(double a, double b) {
        this.a = a;
        this.b = b;
        this.calc = new Calculator();
    }

    @When("^we try to find sum of our numbers")
    public void when() {
        result = calc.sum(a, b);
    }

    @Then("^result should be (\\d)")
    public void then(double res) {
        Assert.assertEquals(res, result, 0.0001);
    }
}
```

Ніяких обмежень щодо кроків для розширення чи на ім’я цих файлів не накладається. Але описане в одному текстовому файлі буде шукатися у всіх java-реалізаціях. Таким чином, дві реалізації визначені описаними кроками недопустимі.

Даний сценарій охоплює лише функцію калькулятора додавання і може бути легко змінений та доповнений, щоб вмістити більше дій для перевірки(ділення, множення, віднімання тощо).

Цей процес можна порівняти із написанням документації до розроблюваного продукту.

Cucumber прекрасно інтегрується в існуючі бібліотеки для запуску тестів, такі як JUnit і TestNG.

Для Cucumber реалізований свій org.junit.runner.Runner:

```
import cucumber.api.junit.Cucumber;
import org.junit.runner.RunWith;
@RunWith (Cucumber.class)
public class CucumberTestRunner {
}
```

У випадку з TestNG, CucumberTestRunner повинен успадковуватися від AbstractTestNGCucumberTests:

```
import cucumber.api.testng.AbstractTestNGCucumberTests;
public class CucumberTests extends AbstractTestNGCucumberTests {
}
```

Cucumber змушує розробників організовувати свої тести у формі BDD, що робить тест ще більш яснішим та зрозумілішим.

Тести, створені в Cucumber, мають тенденцію бути більш інформативними, краще організованими і легше зрозумілими для інших розробників. До того ж вони гарно і структуровано виглядають. Найважливішим є те, що Cucumber перетворив тестування на надзвичайно зручний і корисний досвід.

3.2 Приклад реалізації

Для того, щоб продемонструвати вище зазначений BDD-фреймворк у дії, було використано сайт новин BBC, яким користуються мільйони людей по всьому світу. А саме було перевірено певний набір дій, які роблять повсякденно користувачі даного інтернет-ресурсу.

Якщо бути точнішим, було протестувано основні сторінки даного сайту, а саме: HomePage, SignInPage, NewsPage тощо. В прикладі буде розібране тестування NewsPage. Будуть представлені варіанти тестування без BDD-фреймворків, лише з JUnit, а також із використанням даної технології.

Для того, щоб підключити Cucumber необхідно у pom.xml файл вставити відповідні залежності(dependencies):

```
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-java</artifactId>
    <version>6.6.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>6.6.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.17.0</version>
    <scope>test</scope>
</dependency>
```

Рис. 3.1 Перелік необхідних залежностей

Розглянемо перевірку заголовку головної новини на сайті BBC. Спочатку розглянемо реалізацію даного тесту без використання фреймворку BDD. Сам тест складається з находження головної новини на сторінці “News”,

зберігання його у відповідну змінну “mainTittle”, а потім із порівняння очікуваного результату з актуальним.

```
private static final String TITLE = "Thousands hold 'anti-corona' protests in Berlin";

@Test
public void testMainNewsTitle() {
    String mainTitle = businessLogicLayer.getMainNewsTitle();
    Assert.assertEquals(TITLE, mainTitle,  s2: "Tittle of given main news differ from actual");
}
```

Рис. 3.2 Тест на перевірку заголовку головної новини дня

Тепер перейдемо до розгляду методів які викликаються при кожному запуску даного тесту. Задля приховування бізнес-логіки тесту, ми використаєм клас BusinessLogicLayer. В ньому містяться всі методи, які будуть задіяні при створення кроків певного тесту. В даному випадку ми використаєм із нього getMainNewsTittle(), який складається з виклику методів signIn(), newsButtonClick() та mainNewsTittle().

```
private WebDriver driver;

public BusinessLogicLayer(WebDriver driver) { this.driver = driver; }

public String getMainNewsTitle() {
    return new SignInPage(driver).signIn()
            .newsButtonClick()
            .mainNewsTitle();
}
```

Рис. 3.3 Реалізація getMainNewsTittle()

Тепер же давайте розглянемо реалізацію кожного методу, які викликається в getMainNewsTittle(). Відповідно signIn() відповідає за авторизацію користувача, newsButtonClick() за перехід на сторінку з новинами, а mainNewsTittle() за знаходження заголовку головної новини. Кожен із них розташований у відповідному класі SignInPage, HomePage та NewsPage, які є прототипами певних сторінок сайту BBC.

В класі HomePage розташований метод newsButtonClick(), в якому відбувається клік по кнопці newsButton.

```
@FindBy(xpath = "//nav[@role = 'navigation']//li[@class = 'orb-nav-newsdotcom']/a[contains(text(), 'News')]")
private WebElement newsButton;

public NewsPage newsButtonClick() {
    newsButton.click();
    navLinksWait();
    return new NewsPage(driver);
}
```

Рис. 3.4 HomePage

В класі SignInPage розташований метод signIn(), в якому відбувається клік по кнопці signInPageBtn, введення даних користувача, а саме логін і пароль, та натискання кнопки signInBtn

```
@FindBy(id = "idcta-link")
private WebElement signInPageBtn;

@FindBy(id = "user-identifier-input")
private WebElement login;

@FindBy(id = "password-input")
private WebElement password;

@FindBy(id = "submit-button")
private WebElement signInBtn;

public SignInPage(WebDriver driver) { super(driver); }

public HomePage signIn() {
    signInPageBtn.click();
    signInFieldsWait();
    final String email = "test@gmail.com";
    login.sendKeys(email);
    final String pass = "test12345";
    password.sendKeys(pass);
    signInBtn.click();
    navLinksWait();
    return new HomePage(driver);
}
```

Рис. 3.5 SignInPage

В класі NewsPage розташований метод mainNewsTittle(), в якому відбувається видобуття, за допомого xPath, заголовку головної новини.

```
@FindBy(xpath = "//div[contains(@class, 'primary-item')]//div[contains(@class, 'block@m')]//h3")
private WebElement mainNews;

public String mainNewsTitle() { return mainNews.getText(); }
```

Рис. 3.6 NewsPage

Тепер же розглянемо реалізацію даного тесту з використанням BDD-фреймворку Cucumber.

Спочатку ми створюємо клас CucumberSteps, в якому ми створюємо методи-кроки нашого майбутнього тесту. Кожен новий крок ми позначаємо за допомогою анотацій Given-When-Then. В них ми прописуємо за що відповідає певний метод. Даний тест складається з 3 кроків, а саме:

1 крок – логін користувача;

2 крок – відкриття сторінки “News”(“Новини”);

3 крок – пошук заголовку головної новини дня та порівняння його з очікуваним.

```
@Given("User is signed in")
public void signIn() { new SignInPage(getDriver()).signIn(); }

@When("He opens News Page")
public void openNewsPage() { new HomePage(getDriver()).newsButtonClick(); }

@Then("Check that title {string} of given main news equals actual")
public void checkMainTitle(String string) {
    String mainTitle = new NewsPage(getDriver()).mainNewsTitle();
    assertThat(string)
            .as( description: "Title of given main news differ from actual")
            .isEqualTo(mainTitle);
}
```

Рис. 3.7 Реалізація Cucumber steps

Як було раніше зазначено, тест за допомогою BDD-фреймворку Cucumber має назву story і складається з Cucumber кроків, написаних за

допомогою Gherkin. Нижче наведено приклад сценарію-перевірки, розташованого в директорії test/resources/features даного проекту. Цей файл обов'язково має мати розширення .feature. В ньому ми використовуємо раніше створені кроки із класу CucumberSteps. Також за допомогою Examples ми передаємо очікуваний заголовок.

```
Scenario Outline: Verification that expected main title equals actual
  Given User is signed in
  When He opens News Page
  Then Check that title "<title>" of given main news equals actual
  Examples:
    | title                                    |
    | Thousands hold protests in Berlin |
```

Рис. 3.8 Test scenario

Після цього можна ще додатково створити клас CucumberStepsRunner, за допомогою якого можна запустити всі сценарії, а саме за допомогою анотації CucumberOptions буде просканово всі файли з розширенням .feature в директорії test/resources/features та запущено всі наявні в них тестові-сценарії.

```
@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty"},
        monochrome = true,
        glue = "selenium.bdd",
        features = "src/test/resources/features"
)
public class CucumberStepsRunner {
}
```

Рис. 3.9 CucumberStepsRunner

3.3 Приклад виконання тестів

Для запуску тесту із нашого фреймворку ми використовуємо середовище розробки InteliJ Idea з попередньо встановленими Java 8 та chromedriver (драйвер, за допомогою якою дії описані в тесті виконуються в браузері GoogleChrome).

Під час кожного запуску тесту ми бачимо повідомлення зверху у вікні браузера, що “веб-переглядачем Chrome керує автоматична пробна програма”, тобто усі дії, які відбуваються на екрані є автоматизовані.

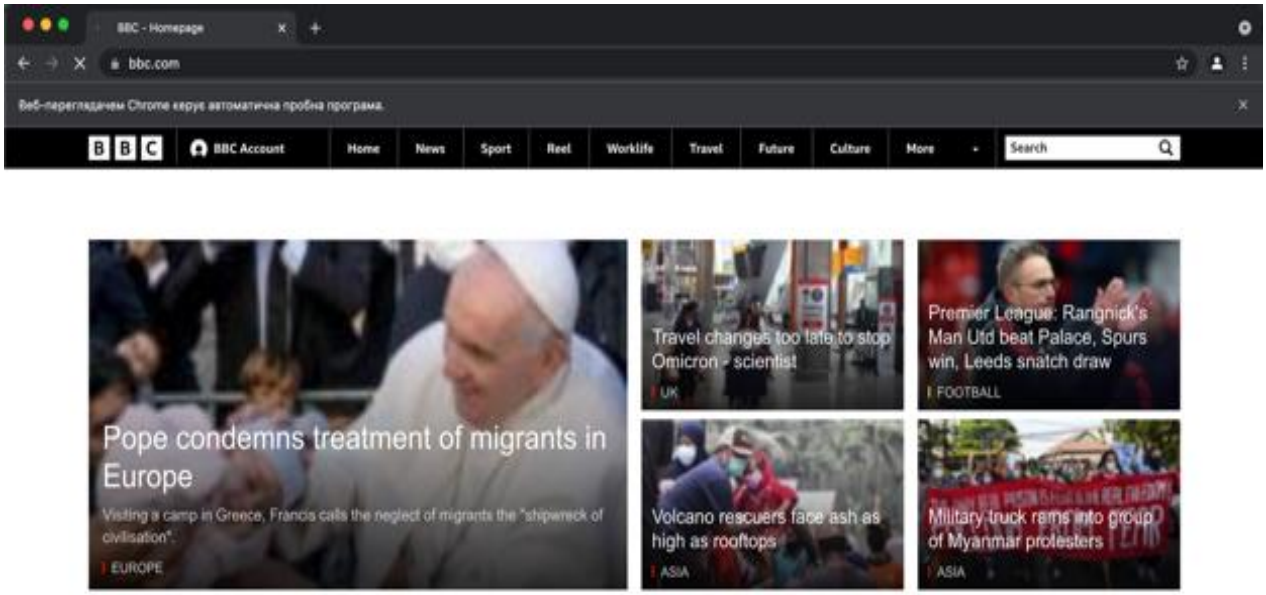


Рис. 3.10 Запуск тесту, головна сторінка сайту BBC

Також, у випадку, якщо ми хочемо перевірити дії, які буде виконувати авторизований користувач, то після відкриття головної сторінки – відбувається перехід на сторінку авторизації, де вводиться логін і пароль користувача.

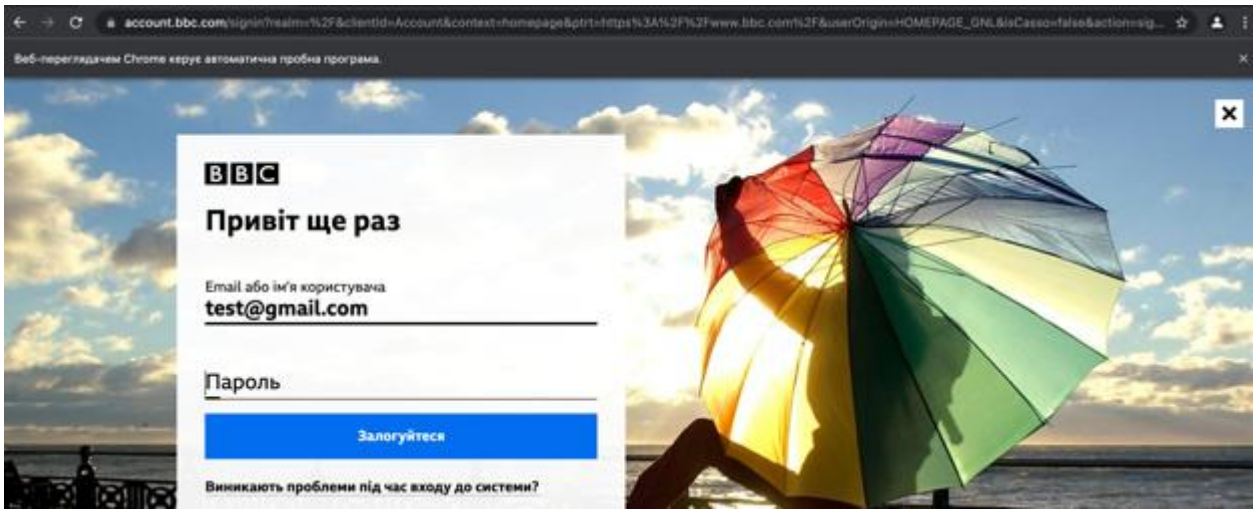


Рис. 3.11 Сторінка авторизації

Після цього в залежності від переслідуваних цілей тесту відбувається перехід на відповідну доступну сторінку даного сайту (NewsPage, MatchesPage тощо).

Розглянемо тест testMainNewsTitle(). Він розпочинається з того, що ми відкриваємо головну сторінку, авторизуємося, потім переходимо на сторінку NewsPage і перевіряємо чи заголовок головної новини дня є таким, яким задумував його редактор сайту.

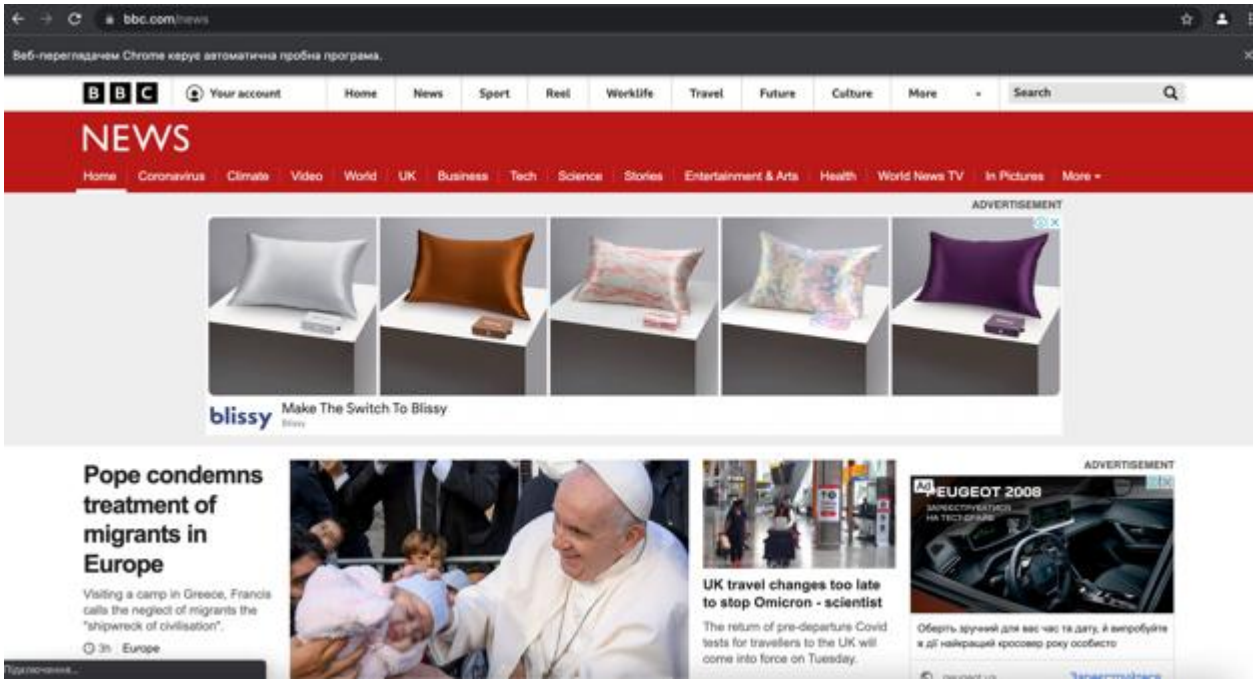


Рис. 3.12 Сторінка NewsPage

Як результат виконання тесту ми матимемо відповідне повідомлення про закінчення тесту у консолі InteliJ Idea. У випадку якщо очікуваний заголовок, який придумав редактор співпадає з тим, що наявний на сайті, то ми отримаємо повідомлення, що тест пройшов і побачимо зелену галочку біля нього.

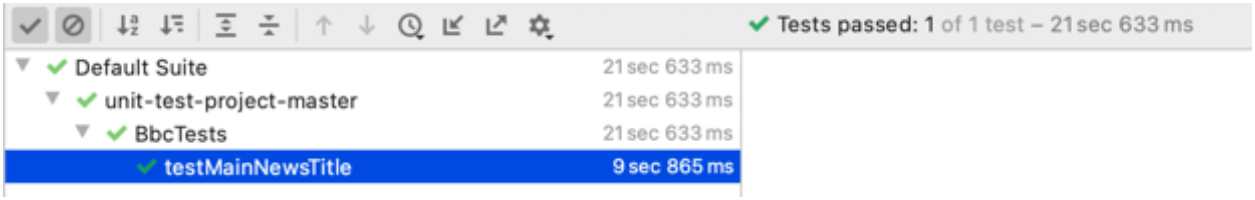


Рис. 3.13 Вдалий запуск тесту

Але якщо щось пішло не так і тест завалився на якомусь певному кроці, то ми отримаємо повідомлення, що тест закінчив свою роботу із помилкою і інформацію про неї.

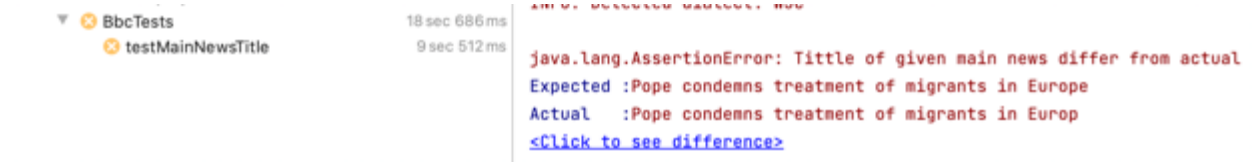


Рис. 3.14 Тест завершився із помилкою

В даному випадку очікуваний заголовок не співпадає з тим, що ми отримали на сторінці NewsPage і як результат тест не пройшов.

Нижче можна розглянути табличку, в якій наведено можливі результати запуску тестів.

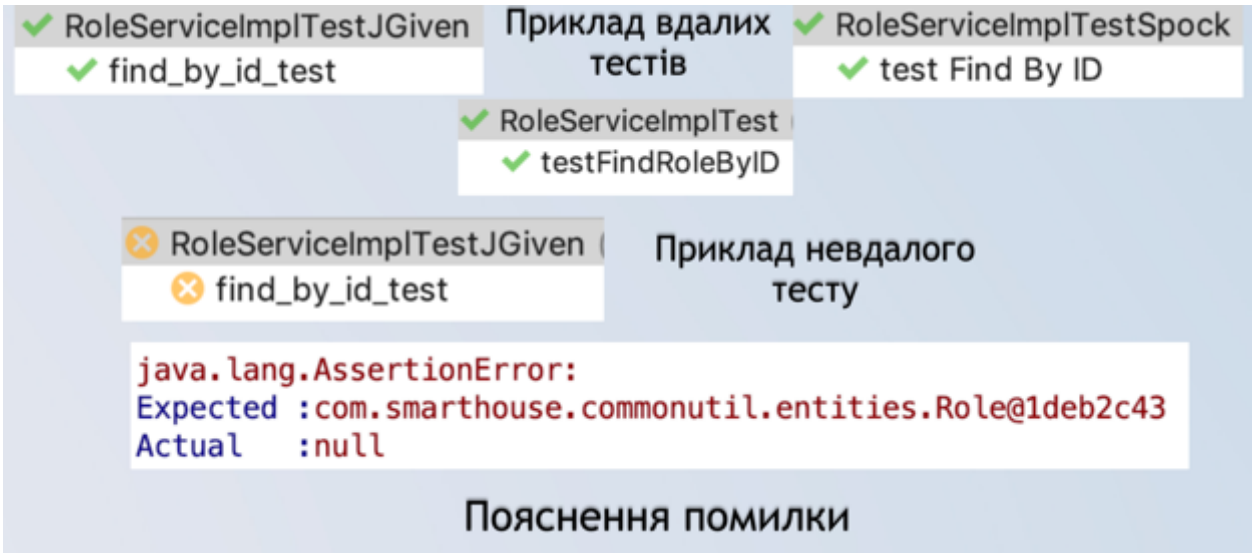


Рис. 3.15 Можливі результати тестів

Крім цього були створені тести для перевірки заголовків другорядних новин із сторінки “Новини”, заголовку новини за регіоном, рахунку певного футбольного матчу, рахунку головного футбольного матчу дня, тощо. Дані тести слугують для того, щоб перевіряти правильну наповненість сайту та відображення певних елементів на ньому. Всіх їх можна знайти в додатку з лістингом програми.

Загалом даний фреймворк дозволяє нам з легкістю організовувати регресійне тестування. Тобто замість того, щоб вручну кожного дня перевіряти чи правильно відображається інформація на сайті, або чи при натисканні на відповідну клавішу відбувається перехід на необхідну сторінку, ми замінюємо всі ці дії за допомогою тестів нашого фреймфорку.

Таким чином, даний тестовий фреймворк, дозволяє нам зекономити витрати на мануальне тестування, і попереджає виникнення помилок у майбутньому, оскільки якщо 1 із тестів, наявних в нашому фреймворку, завершиться із помилкою, то ми дізнаємося про це відразу. Крім цього за допомогою BDD-технології, ми маємо змогу залучити до розробки людей різної спеціалізації і обговорювати з ними проект на “одній мові”.

Висновки до розділу 3

Підсумовуючи все вище сказане у “Розділі 3” можна зробити наступні висновки:

1) основні файли в Cucumber - це текстові файли з розширенням .feature, які містять опис сценаріїв;
2) BDD-сценарії складаються з кроків, реалізація яких знаходиться у відповідно .java файлі (в даному випадки CucumberSteps.java);
3) у Cucumber BDD все, що ви пишете, повинно зводитись до кроків Дано-Коли-Тоді(Given-When-Then), тобто кожен тест можна розділити на три секції;
4) Cucumber вимагає від спеціалістів організовувати свої тести у формі BDD, що робить тест ще більш яснішим;

5) розроблений тестовий фреймворк для перевірки правильності функціонування сайту новин BBC є розширюваним. Тобто ми маємо змогу з легкістю додавати нові кроки, сценарії з метою протестувати наш застосунок якнайкраще;
6) процес тестування з BDD фреймворком Cucumber робить тести більш структурованими і зрозумілішими. Таким чином людина, яка вперше їх побачить, зможе зрозуміти як вони влаштовані більш швидше;
7) при використанні BDD-методологій, тестувальник не відкладає тестування до написання розробником коду. Він може починати роботу зі створення файлів-сценаріїв під час розробки, таким чином зберігаючи час для написання тестів у майбутньому.

Загальні висновки

У даній роботі був проведений аналіз поведінково-орієнтовного тестування веб-застосунків. Були детально проаналізовані методи тестування, а також розглянута технологія BDD.

У вільному доступі існує багато проектів, де застосовується BDD, є документація та роботи. Це є доказом того, що BDD має широкий потенціал використовуватись надалі при тестуванні. Крім того, дана технологія дуже полегшує взаємодію між замовником та виконавцем. Використовуючи її, вони будуть розмовляти на “одній” мові, тому що більшість сценаріїв для тестування пишеться на чистій англійській мові.

Таким чином, використання BDD значно економить час і робота може виконуватись значно швидше, бо замовник може сам проаналізувати, що йому потрібно, а спеціалісту лише доведеться перенести слова замовника на необхідну мову програмування.

В ході дослідження було вивчено літературу та різноманітні інтернет-ресурси, щодо сучасних методів тестування, а також проаналізовані всі плюси та недоліки використання BDD. Також було проведено порівняння за функціоналом різноманітних фреймворків, які реалізують дану технологію. Результатом дослідження є практична робота, в якій продемонстровано тестування веб-застосунка з використанням BDD-фреймворків, а також без них.

Список джерел

1. [Електронний ресурс] стаття “Що таке тестування програмного забезпечення та яке його значення?” https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/
2. [Електронний ресурс] лекція “Тестування програмного продукту” http://lib.mdpu.org.ua/e-book/vstup/L11.htm
3. [Електронний ресурс] Basics of BDD intesting, by Alex McPeak https://crossbrowsertesting.com/blog/development/what-is-bdd-cucumber/
4. [Електронний ресурс] The Pros and Cons of Behavior-Driven Development, by EmilieMaxie https://www.verypossible.com/blog/the-pros-and-cons-of-behavior-driven-development
5. [Електронний ресурс] What is BDD? An Introduction to Behavioral Driven Development, byJithinNair https://blog.testlodge.com/what-is-bdd/
6. [Електронний ресурс] Writing BDD Test Scenarios, by Richard Holmes https://www.departmentofproduct.com/blog/writing-bdd-test-scenarios/
7. [Електронний ресурс] BDD Guidelines and Best Practices, by Amir Ghahrai https://www.testingexcellence.com/bdd-guidelines-best-practices/
8. [Електронний ресурс] Behavior Driven Development (BDD) and Acceptance Testing DrivenDevelopment (ATDD) Tools http://www.agilesoftwaretools.com/behaviordrivendevelopment.php

9. [Електронний ресурс] 8 Best Behavior Driven Development (BDD) Tools and Testing Frameworks https://www.softwaretestinghelp.com/behavior-driven-development-bdd-tools/
10. [Електронний ресурс] JBehaveVsCucumber JVM: Comparison and ExperienceSharing, byThilinaAshenGamage https://medium.com/agile-vision/jbehave-vs-cucumber-jvm-comparison-and-experience-sharing-439dfdf5922d
11. [Електронний ресурс] BDD And Cucumber Tutorial WithExamples https://www.softwaretestinghelp.com/cucumber-bdd-tutorial/
12. [Електронний ресурс] What is Cucumber Testing Tool? Framework Introduction https://www.guru99.com/introduction-to-cucumber.html

ДОДАТОК 1

Лістинг програми

1. Приклад реалізації сторінок HomePage, NewsPage, SignInPage, SportPage, MatchesPage

HomePage

```
package selenium.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class HomePage extends BasePage {
    @FindBy(xpath = "//nav[@role = 'navigation']//li[@class = 'orb-nav-newsdotcom']/a[contains(text(), 'News')]")
    private WebElement newsButton;

    @FindBy(xpath = "//nav[@role = 'navigation']//li[@class = 'orb-nav-sport']/a[text() ='Sport']")
    private WebElement sportsButton;

    public HomePage(WebDriver driver) {
        super(driver);
    }

    public NewsPage newsButtonClick() {
        newsButton.click();
        navLinksWait();
        return new NewsPage(driver);
    }


    public SportPage sportsButtonClick() {
        sportsButton.click();
        navLinksWait();
        return new SportPage(driver);
    }
}
```

SignInPage

```
package selenium.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;

public class SignInPage extends BasePage {

    @FindBy(id = "idcta-link")
    private WebElement signInPageBtn;

    @FindBy(id = "user-identifier-input")
    private WebElement login;

    @FindBy(id = "password-input")
    private WebElement password;

    @FindBy(id = "submit-button")
    private WebElement signInBtn;

    public SignInPage(WebDriver driver) {
        super(driver);
    }

    public HomePage signIn() {
        signInPageBtn.click();
        signInFieldsWait();
        final String email = "test@gmail.com";
        login.sendKeys(email);
        final String pass = "test12345";
        password.sendKeys(pass);
        signInBtn.click();
        navLinksWait();
        return new HomePage(driver);
    }
}
```

NewsPage

```
package selenium.pages;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import java.util.ArrayList;
import java.util.List;

public class NewsPage extends BasePage {
   @FindBy(xpath = "//div[contains(@class, 'primary-
item')]//div[contains(@class, 'block@m')]//h3")
   private WebElement mainNews;
   @FindBy(xpath = "//div[contains(@class, 'primary-
item')]//div[contains(@class, 'block@m')]//li/a")
   private WebElement mainNewsLocationLink;
   @FindBy(id = "site-container")
   private WebElement allNews;
   @FindBy(id = "orb-search-q")
   private WebElement searchField;
   @FindBy(id = "orb-search-button")
   private WebElement searchBtn;

   public NewsPage(WebDriver driver) {
      super(driver); }
   public String mainNewsTitle() {
      return mainNews.getText(); }
   public ArrayList<String> getSecondaryNewsTitles() {
      ArrayList<String> res = new ArrayList<>();
      List<WebElement> titles =
allNews.findElements(By.xpath("//div[contains(@class, 'secondary-
item')]//h3"));
      for (WebElement webElement : titles) {
         String title = webElement.getText();
         res.add(title); }
      return res;
   }
   public MainTitleRegionNews openMainTitleRegionNews() {
      driver.get(mainNewsLocationLink.getAttribute("href"));
      regionMainNewsWait();
      return new MainTitleRegionNews(driver)
}
}
```

SportPage

```
package selenium.pages;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;


public class SportPage extends BasePage {
   @FindBy(xpath = "//div[@role='menubar']//a[@data-stat-title='Football']")
   private WebElement footballTab;

   @FindBy(xpath = "//ul[@role='menu']//a[@data-stat-title='Scores &
Fixtures']")
   private WebElement scoresFixturesTab;


   public SportPage(WebDriver driver) {
      super(driver);
   }


   public MatchesPage footballButtonClick() {
      footballTab.click();
      navLinksWait();
      scoresFixturesTab.click();
      sliderWait();
      return new MatchesPage(driver);
   }
}
```

MatchesPage

```
package selenium.pages;

import org.openqa.selenium.By;
import org.openqa.selenium.Keys;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.support.FindBy;
import selenium.utilities.Score;

import java.util.List;

public class MatchesPage extends BasePage {
    @FindBy(id = "downshift-0-input")
    private WebElement championshipSearchField;

    public MatchesPage(WebDriver driver) {
        super(driver);
    }

    public MatchesPage sendParamsToSearchField(String value) {
        championshipSearchField.sendKeys(value);
        championshipSearchField.sendKeys(Keys.DOWN);
        championshipSearchField.sendKeys(Keys.RETURN);
        sliderWait();
        return new MatchesPage(driver);
    }

    public MatchesPage monthChoose(String testMonth, String year) {
        List<WebElement> months =
driver.findElements(By.xpath("//a[@class='sp-c-date-picker-timeline__item-
inner']"));
        String month;
        for (WebElement webElement : months) {
            month = webElement.getText();
            if (month.contains(testMonth) && month.contains(year)) {
                webElement.click();
                break;
            }
        }
        matchesListWait();
        return new MatchesPage(driver);
```

```
    }

    public CurrentMatchPage openMatch(String teamOne, String teamTwo) {
        List<WebElement> teams = driver.findElements(By.xpath("//article[@class
= 'sp-c-fixture']"));
        for (int i = 0; i < teams.size(); i++) {
            if (teams.get(i).getText().contains(teamOne) &&
teams.get(i).getText().contains(teamTwo)) {
                teams.get(i).click();
                break;
            }
        }
        matchScoreWait();
        return new CurrentMatchPage(driver);
    }

    public Score getScore(String teamOne, String teamTwo) {
        Score res = null;
        List<WebElement> teams = driver.findElements(By.xpath("//article[@class
= 'sp-c-fixture']"));
        List<WebElement> teamOneGoals =
driver.findElements(By.xpath("//span[contains(@class, 'fixture__number--
home')]"));
        List<WebElement> teamTwoGoals =
driver.findElements(By.xpath("//span[contains(@class, 'fixture__number--
away')]"));
        for (int i = 0; i < teams.size(); i++) {
            if (teams.get(i).getText().contains(teamOne) &&
teams.get(i).getText().contains(teamTwo)) {
                res = new Score(Integer.parseInt(teamOneGoals.get(i).getText()),
Integer.parseInt(teamTwoGoals.get(i).getText()));
                break;
            }
        }
        return res;
    }
}
```

2. Приклад реалізації BusinessLogicLayer

```
package selenium.testng.layers;

import org.openqa.selenium.WebDriver;
import selenium.pages.MatchesPage;
import selenium.pages.SignInPage;
import selenium.utilities.Score;

import java.util.Dictionary;


public class BusinessLogicLayer {

    private WebDriver driver;

    public BusinessLogicLayer(WebDriver driver) {
        this.driver = driver;
    }

//метод який повертає заголовок головної новини

    public String getMainNewsTitle() {
        return new SignInPage(driver).signIn()
                .newsButtonClick()
                .mainNewsTitle();
    }

   //метод який повертає заголовки другорядних новин

    public String[] getSecondaryNewsTitles() {
        return new SignInPage(driver).signIn()
                .newsButtonClick()
                .getSecondaryNewsTitles()
                .toArray(new String[0]);
    }

   //метод який повертає заголовок головної новини регіону

    public String getTitleOfMainNewsByRegion() {
        return new SignInPage(driver).signIn()
                .newsButtonClick()
                .openMainTitleRegionNews()
```

```
                .getTitleOfLocationArticle();
    }

//метод який повертає рахунок обраного матчу

    public Score matchScoreOfTheChosenMatch(String championship, String
month, String year, String teamOne, String teamTwo) {
        return new SignInPage(driver).signIn()
                .sportsButtonClick()
                .footballButtonClick()
                .sendParamsToSearchField(championship)
                .monthChoose(month, year)
                .getScore(teamOne, teamTwo);
    }

//метод який повертає рахунок матчу зі сторінки матчів

    public Score getMatchScoreFromMatchPage(String teamOne, String teamTwo)
{
        return new MatchesPage(driver)
                .openMatch(teamOne, teamTwo)
                .getScoreFromMatchPage();
    }

}
```

3. Приклад тестування з JUnit

```
package selenium.testng.tests;

import org.testng.Assert;
import org.testng.annotations.Test;
import selenium.utilities.DictionaryCreator;
import selenium.utilities.Score;

import java.util.Dictionary;

public class BbcTests extends CommonConditions {
   private static final Dictionary<String, String> DICTIONARY =
DictionaryCreator.dictionaryCreate("story",
         "name", "email@@gmail,com",
         "12345", "Ukraine", "true", "false", "true");
   private static final String TITLE = "Thousands hold protests in Berlin";
   private static final String[] SECONDARY_TITLES = {
         "'A true superhero' - Chadwick Boseman remembered",
         "How Black Panther inspired children - and adults", "'Change is slow
in America'",
         "Elon Musk unveils pig with chip in its brain",
         "Teen billed for police overtime after BLM rally"
   };
   private static final Score SCORE = new Score(2, 0);

//тест на перевірку заголовку головної новини

   @Test
   public void testMainNewsTitle() {
      String mainTitle = businessLogicLayer.getMainNewsTitle();
      Assert.assertEquals(TITLE, mainTitle, "Tittle of given main news differ
from actual");
   }

//тест на перевірку заголовків другорядних новин

   @Test
   public void testSecondaryTitles() {
      String[] secondaryTitles = businessLogicLayer.getSecondaryNewsTitles();
      Assert.assertEquals(SECONDARY_TITLES, secondaryTitles, "Tittle of
given secondary news differ " +
            "from actual");
```

```
    }

//тест на перевірку заголовків головної новини регіону

    @Test
    public void testRegionMainTitle() {
        String regionTitle = businessLogicLayer.getTitleOfMainNewsByRegion();
        Assert.assertEquals(TITLE, regionTitle, "Tittle of main news by region
differ from actual");
    }

//тест на перевірку рахунку певного матчу

    @Test
    public void testScoresOfTeams() {
        final String month = "OCT";
        final String year = "2019";
        final String championship = "Scottish Championship";
        final String teamOne = "Dunfermline";
        final String teamTwo = "Arbroath";
        Score matchScoreOfTheChosenMatch =
businessLogicLayer.matchScoreOfTheChosenMatch(championship,
                month, year, teamOne, teamTwo);
        Assert.assertEquals(matchScoreOfTheChosenMatch.matchScore(),
SCORE.matchScore(), "Given match score " +
                "differ from actual on Matches page");
        Score matchScoreFromMatchPage =
businessLogicLayer.getMatchScoreFromMatchPage(teamOne, teamTwo);
        Assert.assertEquals(matchScoreFromMatchPage.matchScore(),
SCORE.matchScore(), "Given match score differ " +
                "from actual on CurrentMatch page");
    }
}
```

4. Реалізація кроків тесту з використанням Gherkin та Cucumber

```
package selenium.bdd.steps;

import io.cucumber.java.en.And;
import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import selenium.pages.*;
import selenium.utilities.Score;

import java.util.Dictionary;
import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;
import static selenium.bdd.driver.DriverHelper.getDriver;
import static selenium.utilities.DictionaryCreator.dictionaryCreate;

public class CucumberSteps {
    private static final Dictionary<String, String> DICTIONARY =
dictionaryCreate("story", "name",
            "email@@gmail,com",
            "12345", "Ukraine", "true", "false", "true");

    @Given("User is signed in")
    public void signIn() {
        new SignInPage(getDriver()).signIn();
    }

    @When("He opens News Page")
    public void openNewsPage() {
        new HomePage(getDriver()).newsButtonClick();
    }

    @And("Clicks region of main news")
    public void openNewsFromRegionOfMainNews() {
        new NewsPage(getDriver()).openMainTitleRegionNews();
    }

    @When("He opens Sport page")
    public void openSportsPage() {
        new HomePage(getDriver()).sportsButtonClick();
    }
```

```
    @And("Opens Football matches page")
    public void openAllFootballMatchesPage() {
        new SportPage(getDriver()).footballButtonClick();
    }

    @And("Chooses {string}")
    public void chooseChampionship(String section) {
        new MatchesPage(getDriver()).sendParamsToSearchField(section);
    }

    @And("Chooses {string} - {string} date")
    public void chooseMonth(String section, String year) {
        new MatchesPage(getDriver()).monthChoose(section, year);
    }

    @When("User opens match {string} - {string}")
    public void openMatch(String teamOne, String teamTwo) {
        new MatchesPage(getDriver()).openMatch(teamOne, teamTwo);
    }

    @Then("Check that title {string} of given main news equals actual")
    public void checkMainTitle(String string) {
        String mainTitle = new NewsPage(getDriver()).mainNewsTitle();
        assertThat(string)
                .as("Title of given main news differ from actual")
                .isEqualTo(mainTitle);
    }

    @Then("Check that tittles of given secondary news equals actual")
    public void checkSecondaryTitles(List<String> expectedSecondaryTitles) {
        List<String> secondaryTitles = new
NewsPage(getDriver()).getSecondaryNewsTitles();
        assertThat(expectedSecondaryTitles)
                .as("Title of given secondary news differ from actual")
                .containsAll(secondaryTitles);
    }

    @Then("Check that title {string} of main news by region equals actual")
    public void checkTitleByRegion(String string) {
        String regionTitle = new
MainTitleRegionNews(getDriver()).getTitleOfLocationArticle();
        assertThat(string)
                .as("Title of main news by region differ from actual")
                .isEqualTo(regionTitle);
    }
```

```
   @Then("Check that match {string} - {string} has score {int} - {int} on
Matches page")
   public void checkMatchScoreFromMatchPage(String teamOne, String
teamTwo, int goalsOne, int goalsTwo) {
       Score matchScoreOfTheMatch = new
MatchesPage(getDriver()).getScore(teamOne, teamTwo);
       assertThat(new Score(goalsOne, goalsTwo).matchScore())
             .as("Given match score differ from actual on Matches page")
             .isEqualTo(matchScoreOfTheMatch.matchScore());
   }

   @Then("Check that given match has score {int} - {int} on CurrentMatch
page")
   public void checkMatchScoreFromCurrentMatchPage(int goalsOne, int
goalsTwo) {
       Score matchScoreFromMatchPage = new
CurrentMatchPage(getDriver()).getScoreFromMatchPage();
       assertThat(new Score(goalsOne, goalsTwo).matchScore())
             .as("Given match score differ from actual on CurrentMatch page")
             .isEqualTo(matchScoreFromMatchPage.matchScore());
   }

}
```

5. Приклад створених тестових сценаріїв-історій з Cucumber

Feature: As a tester I want to implement my tasks using BDD So that I can complete my project
  Scenario Outline: Verification that expected main title equals actual
    Given User is signed in
    When He opens News Page
    Then Check that title "<title>" of given main news equals actual
    Examples:
      | *title* |
      | **Thousands hold protests in Berlin** |

  Scenario: Verification that expected secondary titles equal actual
    Given User is signed in
    When He opens News Page
    Then Check that tittles of given secondary news equals actual
      | **'A true superhero' - Chadwick Boseman remembered** |
      | **How Black Panther inspired children - and adults** |
      | **'Change is slow in America'** |
      | **Elon Musk unveils pig with chip in its brain** |
      | **Teen billed for police overtime after BLM rally** |

  Scenario Outline: Verification that expected main news will be main news in its region
    Given User is signed in
    When He opens News Page
    And Clicks region of main news
    Then Check that title "<title>" of main news by region equals actual
    Examples:
      | *title* |
      | **Thousands hold protests in Berlin** |

  Scenario: Verification that correct match score of chosen match is displayed
    Given User is signed in
    When He opens Sport page
    And Opens Football matches page
    And Chooses "Scottish Championship"
    And Chooses "OCT" - "2019" date
    Then Check that match "Dunfermline" - "Arbroath" has score 2 - 0 on Matches page
    When User opens match "Dunfermline" - "Arbroath"
    Then Check that given match has score 2 - 0 on CurrentMatch page

6. CucumberStepsRunner – клас для запуску всіх навних BDD-сценаріїв

```
package selenium.bdd.runner;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty"},
        monochrome = true,
        glue = "selenium.bdd",
        features = "src/test/resources/features"
)
public class CucumberStepsRunner {
}
```

ДОДАТОК 2

Копії публікацій за темою магістерської роботи

РЕЗАНОВА В.Г., НІКІТЧЕНКО Я.Ю.

СТВОРЕННЯ ФРЕЙМВОРКУ ДЛЯ АВТОМАТИЗОВАНОГО ТЕСТУВАННЯ ВЕБ-ЗАСТОСУНКУ З ВИКОРИСТАННЯМ BDD-ТЕХНОЛОГІЙ

REZANOVA V.G., NIKITCHENKO Y.Y.

CREATION OF A FRAMEWORK FOR AUTOMATED TESTING OF A WEB APPLICATION USING BDD TECHNOLOGIES

Purpose and tasks. The aim of the work is to create a framework for automated testing of a web application using BDD technologies

The task is to automate all processes of checking criteria to make process of testing cheaper and faster.

Object and subject of research. The object of the research is automation and programming of the testing framework.

The subject of the study is is the process of creating and implementing testing steps using BDD-technology to check that all criteria was met.

Вступ

Тестування — це широкий процес, який складається з декількох взаємопов’язаних процесів. Іншими словами сукупності процесів.

Часто говорячи про тестування, люди уявляють собі картинку, в якій спеціаліст перевіряє чи за всіма параметрами програма працює ідеально. Але якість не є абсолютною, це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність програмного забезпечення. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Існує багато підходів до тестування програмного забезпечення, але ефективне тестування складних продуктів - це по суті дослідницький процес, а не тільки створення і виконання рутинної процедури.

Тестування пронизує весь життєвий цикл ПЗ, починаючи від проектування і закінчуючи невизначено довгим етапом експлуатації. Ці роботи безпосередньо пов'язані із завданнями управління вимогами та змінами, адже метою тестування є якраз можливість переконатися у відповідності програм заявленим вимогам.

Тестування - процес також ітераційний. Після виявлення та виправлення кожної помилки обов'язково слід повторити тести, щоб переконатися у працездатності програми. Більше того, для ідентифікації причини виявленої проблеми може знадобитися проведення спеціальної додаткової перевірки. При цьому потрібно завжди пам'ятати про фундаментальний висновок, зроблений професором Едсжером Дейкстри у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!".

Постановка завдання

Тестування програмного забезпечення є невід'ємною частиною створення програмного продукту. Від того, наскільки досконало проведені тести, залежить те, як скоро проект буде зданий остаточно, і чи буде необхідність згодом усувати помилки.

Люди схильні помилятися, людські помилки можуть призводити до порушення нормальної роботи програмного забезпечення на всіх стадіях розробки, причому наслідки цього можуть бути найрізноманітнішими — від незначних до катастрофічних.

Отже мета і завдання моєї роботи - це розглянути різноманітні підходи до тестування програмного забезпечення, розглянути особливості BDD, а також створити фреймворк для автоматизованого тестування.

Основна частина

У залежності від переслідуваних цілей види тестування можна умовно розділити на наступні типи:

1. Функціональні.
2. Нефункціональні.
3. Пов'язані зі змінами.

Сучасні проекти все частіше висувають високі вимоги до покриття автоматичними тестами. У наш час писати тести не просто ознака хорошого тону, але одна з вимог, яка пред'являється до коду. Все частіше ми чуємо такі абревіатури, як TDD (TestDrivenDevelopment) і BDD (BehaviourDrivenDevelopment) і багато строго слідують цим підходам в розробці.

Основною ідеєю даної методології є поєднання в процесі розробки чисто технічних інтересів і інтересів бізнесу, дозволяючи тим самим керуючому персоналу і програмістам говорити на одній мові.

Для спілкування між цими групами персоналу використовується предметно-орієнтована мова, основу якої становлять конструкції з природної мови, зрозумілі неспеціалісту, яка зазвичай виражає поведінку програмного продукту і очікувані результати.

```
Scenario Outline: Verification that expected title will be title of main news on News Page
  Given User is signed in
  When He opens News Page
  Then Check that title "<title>" of main news equals actual
  Examples:
    | title                                  |
    | Ikea France fined €1m for snooping on staff |
```

Рис. 1 Приклад сценарію BDD

Проте за кожним словом даного сценарію скривається певна функція, яка на виконує певні кроки, а вкінці порівнює отриманий результат з очікуваним.

```
@Given("User is signed in")
public void signIn() { new SignInPage(getDriver()).signIn(); }

@When("He opens News Page")
public void openNewsPage() { new HomePage(getDriver()).newsButtonClick(); }

@Then("Check that title {string} of main news equals actual")
public void checkMainTitle(String string) {
    String mainTitle = new NewsPage(getDriver()).mainNewsTitle();
    assertThat(string)
            .as( description: "Title of given main news differ from actual")
            .isEqualTo(mainTitle);
}
```

Рис. 2 Методи, які виконують кроки сценарію

Висновки

Проведення перевірки якості продукту є невід'ємною частиною розробки програмного забезпечення. Тестування не тільки дозволяє перевірити, чи всі вимоги до проекту виконані, а й допомагає у виявленні помилок та дефектів, що пом'якшує наслідки і ризики втрат.

У вільному доступі існує багато проектів, де застосовується BDD, є документація та роботи. Це є доказом того, що BDD має широкий потенціал використовуватись надалі при тестуванні. Крім того, дана технологія дуже полегшує взаємодію між замовником та виконавцем. Використовуючи її, вони будуть розмовляти на "одній" мові, тому що більшість сценаріїв для тестування пишеться на чистій англійській мові.

Використання BDD значно економить час і робота може виконуватись значно швидше, бо замовник може сам проаналізувати, що йому потрібно, а спеціалісту лише доведеться перенести слова замовника на необхідну мову програмування.

Ключові слова: тестування, процес розробки програмного забезпечення, Behavior-DrivenDevelopment.

Література

1. [Електронний ресурс] стаття "Що таке тестування програмного забезпечення та яке його значення?" https://www.quality-assurance-group.com/shho-take-testuvannya-programnogo-zabezpechennya-ta-yake-jogo-znachennya/
2. [Електронний ресурс] лекція "Тестування програмного продукту" http://lib.mdpu.org.ua/e-book/vstup/L11.htm
3. [Електронний ресурс] Basics of BDD in testing, byAlexMcPeakhttps://crossbrowsertesting.com/blog/development/what-is-bdd-cucumber/
4. [Електронний ресурс] ThePros and Cons of Behavior-Driven Development, by Emilie Maxie https://www.verypossible.com/blog/the-pros-and-cons-of-behavior-driven-development

Рис. 1 Зміст "Інформаційні технології в науці, виробництві та підприємництві"

Рис. 2 Обкладинка ""Інформаційні технології в науці, виробництві та підприємництві""

Міністерство освіти і науки України

Київський національний університет технологій та дизайну

МЕХАТРОННІ СИСТЕМИ: ІННОВАЦІЇ ТА ІНЖИНІРИНГ

ТЕЗИ ДОПОВІДЕЙ

V МІЖНАРОДНОЇ НАУКОВО-ПРАКТИЧНОЇ КОНФЕРЕНЦІЇ

4 листопада 2021

MSIE 2021

Київ 2021

Рис. 3 Обкладинка Тез доповідей з 5 Міжнародної науково-практичної конференції

Рис. 4 Зміст Тез доповідей з 5 Міжнародної науково-практичної конференції

УДК 677.072.6

АВТОМАТИЗОВАНЕ ДОСЛІДЖЕННЯ РЕОЛОГІЧНИХ ВЛАСТИВОСТЕЙ МОДИФІКОВАНИХ ПОЛІМЕРІВ

В.Г. Резанова, к.т.н., доцент,
Київський національний університет технологій та дизайну
М.П. Ніка, магістрант
Київський національний університет технологій та дизайну
Я.Ю. Нікітченко, магістрант
Київський національний університет технологій та дизайну

Ключові слова: програмне забезпечення, розплав полімерів, в'язкість.

Дослідження явищ структуроутворення має великий науковий інтерес з точки зору створення загальної теорії процесів переробки сумішей полімерів, визначення ролі входових процесів, які відіграють вирішальну роль не тільки при переробці розплавів сумішей, але й при переробці розплавів індивідуальних полімерів [1]. Вивчення механізмів,процесів та явищ, що спостерігаються при переробці розплавів сумішей полімерів, є важливим і актуальним та підлягає подальшому дослідженню.

Розрахунок параметрів течії розплавів полімерів та представлення одержаних результатів у графічному вигляді є достатньо трудомістким та потребує значних затрат часу. Виходячи із вищесказаного, в роботі ставиться актуальна задача дослідження, розрахунку та зручного графічного представлення реологічних характеристик дисперсійних середовищ.

Особливістю переробки полімерів у вироби є необхідність їх переводу у в'язко-текучий стан з метою надання необхідної форми. Відомо, що в основі класичної гідромеханіки лежить модель в'язкої рідини Ньютона, згідно з якою напруга зсуву (□) прямо пропорційна швидкості деформації (□): □=□□, де коефіцієнт пропорційності □ називають в'язкістю. Характер течії високомолекулярних сполук підпорядковується ступеневому закону: □=□□^{n} де n – ступінь відхилення від ньютонівської течії. З огляду на це, вивченого реологічну поведінку вихідних та модифікованих розплавів полімерів з метою встановлення основних закономірностей їх течії як чинника, що впливає на технологічні параметри переробки. Програмне забезпечення розробляли в середовищі Delphi мовою Object Pascal [2, 3].

Реологічні характеристики досліджуваних розплавів полімерних систем вивчали за допомогою капілярного віскозиметра. Течія розплаву через капіляр відбувається за рахунок перепаду тисків між його кінцями. Обробку експериментальних результатів здійснювали з використанням загальноприйнятої методики для неньютонівських систем. Напругу зсуву на стінці капіляру визначали за співвідношенням: $\square = \frac{4r \square P}{\square \square d_n^2 \square 2L} = K_1 \square P$, де

r, L – радіус і довжина капіляру відповідно; d_n – діаметр поршня; K_1 – постійна величина для даного капіляру, яка залежить від його діаметра і довжини. За отриманими даними будується попередня крива течії, що зв'язує напругу із градієнтом швидкості зсуву на стінці капіляру. З неї розраховують режим течії як тангенс кута нахилу дотичної в даній точці кривої: $n = \frac{\Delta \lg D}{\Delta \lg \Delta}$.

Розроблене програмне забезпечення дозволяє представити у графічному вигляді істинну криву течії $\lg \Delta = f(\lg \Delta)$, а також залежності η від напруги і швидкості зсуву. На рис. 1 наведено розраховані значення функції $\lg \Delta = f(\lg \Delta)$ для розплаву поліпропілену, наповненого 1,0 мас. % Ag/Al_2O_3.

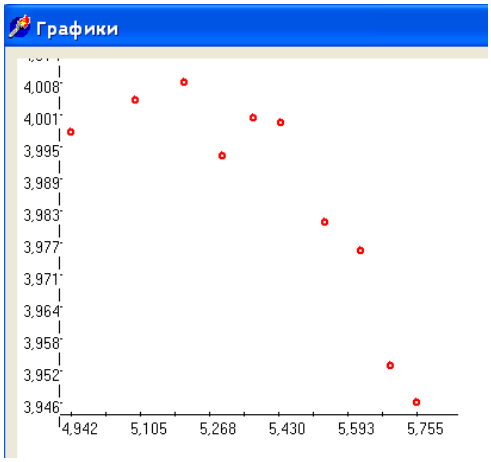


Рисунок 1 - Програмне зображення залежності $\lg \Delta = f(\lg \Delta)$

Створена програма дозволяє розрахувати в'язкість та режим течії розплавів, а також представити результати у вигляді графічних зображень кривої течії і залежності в'язкості від напруги та швидкості зсуву. Програма дозволяє суттєво спростити і зробити набагато ефективнішим процес обробки експериментальних результатів, а також вибрати технологічні параметри переробки в залежності від реологічних характеристик розплавів.

Список використаних джерел

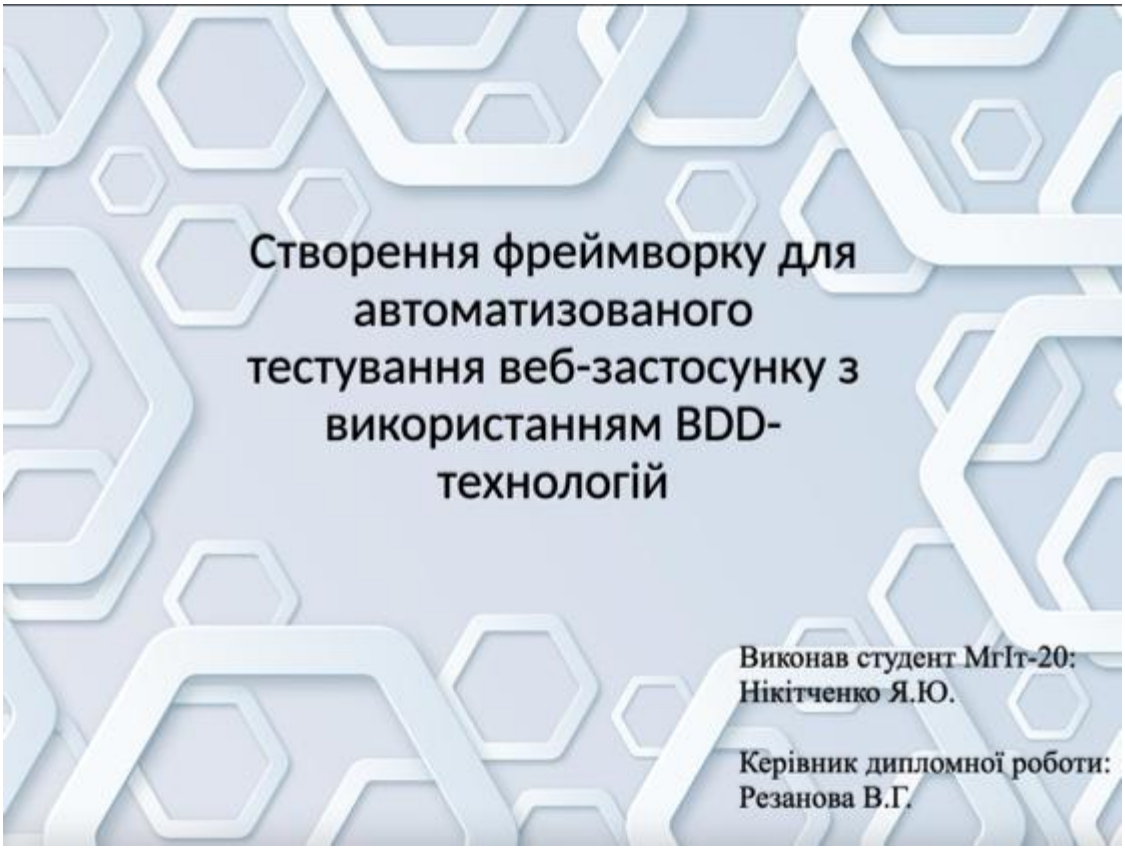
1. Rezanova N.M., Rezanova V.G., Plavan V.P., Viltsaniuk O.O. The influence of nano-additives on the formation of matrix-fibrillar structure in the polymer mixture melts and on the properties of complex threads // Vlákna a textil (Bratislava, Slovak Republic) - №2, 2017. - p. 37-42

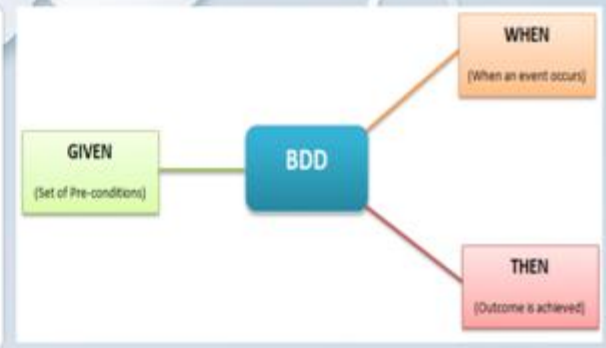
2. Фленов М. Библия Delphi (3-е издание) // СПб.: БХВ-Петербург, 2012 – 688 с.

3. Осипов Д. Л. Delphi. Программирование для Windows, OS X, iOS и Android // СПб.: БХВ-Петербург, - 2014. – 464 с.

ДОДАТОК 3

Презентація дипломної магістерської роботи





Jasmine
Behavior-Driven Javascript

cucumber

easyb

Serenity BDD

jbehave

Concordion

Основа BDD та фреймворки

Опис практичної частини роботи

Використаний фреймворк

```
private static final String TITLE = "Thousands hold 'anti-corona' protests in Berlin";

@Test
public void testMainNewsTitle() {
    String mainTitle = businessLogicLayer.getMainNewsTitle();
    Assert.assertEquals(TITLE, mainTitle,  "Tittle of given main news differ from actual");
}
```

```
private WebDriver driver;

public BusinessLogicLayer(WebDriver driver) { this.driver = driver; }

public String getMainNewsTitle() {
    return new SignInPage(driver).signIn()
            .newsButtonClick()
            .mainNewsTitle();
}
```

JUnit test

Приклад реалізації без BDD-фреймворків

Особливості реалізації з BDD-фреймворками

```
@Given("User is signed in")
public void signIn() { new SignInPage(getDriver()).signIn(); }

@When("He opens News Page")
public void openNewsPage() { new HomePage(getDriver()).newsButtonClick(); }

@Then("Check that title (string) of given main news equals actual")
public void checkMainTitle(String string) {
    String mainTitle = new NewsPage(getDriver()).mainNewsTitle();
    assertThat(string)
            .as( description: "Title of given main news differ from actual")
            .isEqualTo(mainTitle);
}
```

CucumberSteps

```
Scenario Outline: Verification that expected main title equals actual
  Given User is signed in
  When He opens News Page
  Then Check that title "<title>" of given main news equals actual
  Examples:
    | title                                                |
    | Thousands hold protests in Berlin |
```

Cucumber Test scenario

```
@RunWith(Cucumber.class)
@CucumberOptions(
        plugin = {"pretty"},
        monochrome = true,
        glue = "selenium.bdd",
        features = "src/test/resources/features"
)
public class CucumberStepsRunner {
}
```

CucumberSteps Runner

RoleServiceImplTestJGiven
find_by_id_test

Приклад вдалих тестів

RoleServiceImplTestSpock
test Find By ID

RoleServiceImplTest
testFindRoleByID

Приклад невдалого тесту

RoleServiceImplTestJGiven
find_by_id_test


```
java.lang.AssertionError:
Expected :com.smarthouse.commonutil.entities.Role@1deb2c43
Actual   :null
```

Пояснення помилки

Результат виконання тестів

Conclusion

Дякую за увагу

Клацніть, щоб змінити