**Лабораторна робота №07-1.** Підготовка тестів для перевірки ПЗ**.**

**Мета роботи**. Навчитися застосовувати на практиці знання щодо організації проведення тестування програмного продукту.

**Завдання на лабораторну роботу**

1. Ознайомитись з теоретичними відомостями, необхідними для виконання роботи.

2. Для власного проекту потрібно розробити тестовий план (майстер тест-план ) і тест-кейси для перевірки конкретної функціональності програмного застосунку або сайту, а також форму, за якою будуть фіксуватися виявлені в процесі тестування дефекти (приклади тест-плану і тест кейсу надані в кінці ПР).

3. Оформити звіт, де надати у вигляді таблиць тестовий план, тест-кейси, форму для фіксації дефектів, виявлених в процесі тестування, додавши відповідні пояснення щодо обраного рішення.

**Контрольні запитання**

1. Що таке тестування?
2. Що таке відмовостійкість?
3. Які види тестування ви знаєте?
4. В чому полягає суть тестування?
5. Що найчастіше тестується в програмних засобах?
6. В яких моделях немає тестування?
7. Що таке модульне тестування?
8. Що таке інтеграційне тестування?
9. В чому полягає зміст тестування за допомогою «білого» ящику?

Результати надсилати на електронну адресу викладача [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com)у вигляді текстових файлів з іменем у форматі **ОРІ<Номер групи><Номер лекції / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, R - лабораторна]<Прізвище англійською>**. Наприклад, **ОРІ4112R**buts.doc. Відповіді повинні бути не довгими і змістовними. Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності відповідей-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк відсилки ЛР ІПЗ-41 25.09.2024**

**ІПЗ-42**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Тестування - процес виявлення фактів розбіжностей з вимогами (помилок) до програмного продукту. Цей процес полягає у перевірці відповідності програмного продукту чи сайту заявленим характеристикам і вимогам, вимогам експлуатації в різних оточеннях, з різними навантаженнями, вимогам з безпеки, вимогам по ергономіці і зручності використання. Залежно від спрямованості тестування, перевіряється та чи інша особливість програми або веб-сайту. Як правило, процес тестування документується у вигляді тестового плану і тест-кейсів. **Тестовий план** описує стратегію тестування, методи і засоби тестування, порядок та інші його особливості. **Тест-кейси** описують послідовні покрокові операції перевірки функціоналу програми або веб-сайту. Це мінімальні елементарні операції звірки для кожної функції або елемента додатки.

Як правило, на фазі тестування здійснюється і виправлення ідентифікованих помилок, включає:

• локалізацію помилок

• знаходження причин помилок

• коригування програми.

Тестування поділяють на статичне і динамічне:

• Статичне тестування виявляє невірні конструкції або невірні відносини об'єктів програми (помилки формального завдання) формальними методами аналізу без виконання програми, що тестується.

• Динамічне тестування здійснює виявлення помилок під час виконання програми.

Тестування закінчується, коли було виконано або успішно перевірено достатню кількість тестів у відповідності з обраним критерієм тестування.

Тестування вирішує кілька основних завдань:

• дає впевненість у якості кінцевого продукту, підтверджує що всі заявлені функціональні вимоги реалізовані, програмний продукт їм відповідає і не має помилок у програмному коді;

• підтверджує, що програмний продукт здатний виконуватися у всіх заявлених режимах і на всіх підтримуваних ОС або Web-браузерах коректно;

• гарантує, що збережені і оброблені дані надійно захищені від стороннього доступу та "злому";

• визначає, яке максимальне навантаження на сервер, локальну мережу, БД може бути коректно оброблено програмним продуктом;

• дозволяє переконатися в тому, що користувач може "інтуїтивно" використовувати продукт або послугу не плутаючись у складних переплетеннях інтерфейсів.

На ранніх стадіях тестування представляло собою досить примітивний процес, і зовсім не відповідало виконання тих завдань, які були для нього поставлені, а саме: підвищення якості шляхом покриття максимальної функціональності продукту, зменшення часу і зниження витрат на тестування.

На початковому етапі тестування являло собою малозрозумілу область. Воно було лише малооплачуваним доповненням до процесу розробки, стартовою точкою для новачків, які прагнуть підвищити рівень своїх знань і досягти звання розробника. Існуючі інструменти та методи тестування надавали можливість для придбання навичок автоматизації, проте вони були дорогими, складними і неефективними для досягнення цих потреб. На цьому етапі виконавче керівництво не звертало уваги на існування тестування - передбачалося, що цей процес буде відбуватися сам собою.

На наступному етапі прийшло розуміння суті та важливості тестування як окремої частини процесу розробки. Разом з розумінням призначення тестування усіма зацікавленими, виникли і проблеми: як організувати і фінансувати тестування, як визначити його правильний напрямок. Почалося дослідження існуючих інструментів, вивчення яких часто шкодило тестуванню: воно підмінялося безсистемним перебором інструментів без чіткого розуміння, які завдання вони допоможуть вирішити. Окрім того, на цьому етапі розуміння необхідності тестування вищими керівними органами і раніше залишалося на низькому рівні.

Серед ключових моментів цих етапів можна виділити декілька найбільш вагомих. Перший - це брак участі з боку керівництва: без чіткого розуміння необхідності тестування, його місця в процесі розробки і важливості результату, яке дає тестування, його розвиток приречене. Другим важливим моментом є концепція «тестування заради тестування» і вибір інструменту без керівної стратегії вибору. Це подібно будівництві дороги без напрямку: дорога-то буде побудована, але шанси того, що буде використана - мінімальні. Тобто, тестування і вибір інструменту для нього без певної стратегії вибору та використання призведе до того, що всі зусилля для досягнення заявлених цілей процесу будуть зведені нанівець.

Результат перших двох етапів - розробка таких програм тестування, які будуть відповідати вимогам організації до розробки кращого програмного забезпечення більш швидким і дешевим методом.

Рівні тестування

• Модульне тестування.

• Інтеграційне тестування.

• Системне тестування.

Рівні тестування обираються відповідно етапів тестування, які поділяються так:

* тестування компонентів;
* тестування модулів;
* тестування підсистем;
* тестування системи;
* приймальні випробування.

**Модульне тестування**

Модульне тестування - це тестування програми на рівні окремо взятих модулів, функцій або класів. Мета модульного тестування полягає у виявленні локалізованих в модулі помилок у реалізації алгоритмів, а також у визначенні ступеня готовності системи до переходу на наступний рівень розробки і тестування. Модульне тестування проводиться за принципом "білого ящика". Модульне тестування зазвичай передбачає створення навколо кожного модуля певного середовища.

На рівні модульного тестування найпростіше виявити дефекти, пов'язані з алгоритмічними помилками і помилками кодування алгоритмів. Помилки, пов'язані з невірним трактуванням даних, некоректною реалізацією інтерфейсів, сумісністю, продуктивністю тощо, зазвичай виявляються на більш пізніх стадіях тестування.

**Інтеграційне тестування**

Інтеграційне тестування - це тестування частини системи, що складається з двох і більше модулів. Основне завдання інтеграційного тестування - пошук дефектів, пов'язаних з помилками в реалізації та інтерпретації взаємодії між модулями. Так само, як і модульне тестування, цей вид тестування оперує інтерфейсами модулів і підсистем та вимагає створення тестового оточення.

Основна різниця між модульним і інтеграційним тестуванням полягає в типах виявлених дефектів. Зокрема, на рівні інтеграційного тестування часто застосовуються методи, пов'язані з покриттям інтерфейсів. Інтеграційне тестування використовує модель "білого ящика" на модульному рівні.

**Системне тестування**

Основне завдання системного тестування - виявлення дефектів, пов'язаних з роботою системи в цілому:

* невірне використання ресурсів системи;
* непередбачені комбінації даних рівня користувача;
* несумісність з оточенням;
* непередбачені сценарії використання;
* відсутня або невірна функціональність;
* незручність у застосуванні тощо.

Системне тестування проводиться над проектом в цілому за допомогою методу «чорного ящика».

**Тестування «білого ящика» і «чорного ящика»**

«Тестування білого ящика» і «тестування чорного ящика» ставляться до того, чи має розробник тестів доступ до вихідного коду тестованого ПЗ, або ж тестування виконується через інтерфейс користувача або прикладний програмний інтерфейс, наданий через модуль, що тестується.

При тестуванні білого ящика розробник тесту має доступ до вихідного коду і може писати код, який пов'язаний з бібліотеками тестованого ПЗ. Це типово для юніт-тестування (англ. unit testing), при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції до певної міри є працездатні і стійкі,.

При тестуванні чорного ящика тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або через зовнішні інтерфейси, що дозволяють іншому комп'ютеру або іншому процесу підключитися до системи для тестування. Наприклад, тестуючий модуль може віртуально натискати клавіші або кнопки миші в програмі, що тестується за допомогою механізму взаємодії процесів, з упевненістю в тому, чи все йде правильно, що ці події викликають той же відгук, що й реальні натискання клавіш і кнопок миші. Як правило, тестування чорного ящика ведеться з використанням специфікацій або інших документів, що описують вимоги до системи.

**Області еквівалентності**

Вхідні дані програм часто можна розбити на декілька класів. Вхідні дані, що належать одному класу, мають загальні властивості, наприклад це додатні числа, від’ємні числа, рядки без пробілів і тому подібне. Зазвичай для всіх даних з якого-небудь класу поведінка програми однакова (еквівалентна). Через це такі класи даних іноді називають областями еквівалентності. Один з систематичних методів виявлення дефектів полягає у визначенні всіх областей еквівалентності, що обробляються програмою. Контрольні тести розробляються так, щоб вхідні і вихідні дані лежали в межах цих областей.

**Класифікація дефектів.**

***Дефект (баг)*** – це невідповідність фактичного результату виконання програми очікуваному результату. Дефекти виявляються на етапі тестування програмного забезпечення, коли тестер проводить порівняння отриманих результатів роботи програми (компонента або дизайну) з очікуваним результатом, описаним у специфікації вимог.

***Error*** – помилка користувача, тобто він намагається використовувати програму іншим способом, наприклад, уводить літери в поля, де потрібно вводити цифри (вік, кількість товару тощо). У якісній програмі передбачені такі ситуації і видаються повідомлення про помилку (error message), із червоним хрестиком.

***Bug (defect)*** – помилка програміста (або дизайнера або того, хто бере участь у розробці), тобто якщо в програмі, що щось не так, як планувалося, і програма виходить з-під контролю. Наприклад, якщо не контролюється введення користувача, неправильні дані викликають руйнування чи інші проблеми в роботі програми. Або всередині програма побудована так, що спочатку не відповідає тому, що від неї очікується.

***Failure*** – збій (причому не обов’язково апаратний) у роботі компонента, усієї програми або системи. Тобто існують такі дефекти, які призводять до збоїв (A defect caused the failure) і існують такі, які не призводять, наприклад, дефекти інтерфейсу користувача. Але апаратний збій, ніяк не пов’язаний з ПЗ, теж є failure - збій.

***Життєвий цикл дефекту (bug workflow)*** – це послідовність етапів, які проходить дефект на своєму шляху з моменту його створення до остаточного закриття. Для кращого сприйняття зображується у вигляді схеми з можливими статусами і діями, які призводять до зміни цих статусів. Найпростіший життєвий цикл дефекту / бага має такий вигляд:

*Новий* (статус присвоюється автоматично після внесення баг-репорт).

*Відкритий* (баг отримує цей статус після того, як була проведена його валідація керівником команди і ця помилка дійсно має бути виправлена).

*Відхилений* (присвоюється також після аналізу нового бага керівником команди в разі, якщо описана помилка вже раніше була внесена в систему (дублікат) або з якихось причин непотрібне її виправлення).

*Виправлений* (присвоюється фахівцем розробки після того, як помилка була усунена).

*Повторно відкритий* (у разі повторного виникнення помилки після її попереднього виправлення).

*Закритий* (після остаточного виправлення бага і проведення додаткової перевірки).

Окрім вищевказаних основних статусів можуть також додатково використовуватися резолюції, наприклад:

– призначено на …;

– потрібна додаткова інформація;

– у процесі виправлення;

– не може бути відтворений;

– на регресійне тестування.

**Тест-кейс (Test Case)**

Тест-кейс– це тестовий артефакт, суть якого полягає у виконанні певної кількості дій і/або умов, необхідних для перевірки певної функціональності програмної системи, що розробляється. Структура даного артефакту полягає у визначенні:

- що треба зробити (Action) ;

– який очікуваний результат (Expected result);

– який отримано фактичний результат (Test result).

Безпосередньо сам тестовий випадок складається з 3 частин:

1. ***PreConditions*** (передумови) – або список кроків, які призводять систему в стан, придатний для тестування, або список перевірок умов того, що система вже знаходитися в необхідному стані.

2. ***Test Case Description*** (опис тестового випадку) – список дій, за допомогою яких здійснюється основна перевірка функціоналу (після якої і звіряється фактичний результат з очікуваним).

3. ***PostConditions*** (післяумови) – список дій, які повертають систему в початковий стан.

**Тестовий план (Testplan)**

Тест-план або план тестування – документ, що описує весь обсяг робіт з тестування, починаючи з опису тестованих об’єктів, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, оцінки ризиків з варіантами їх вирішення.

Тест-план є важливою складовою будь-якого процесу тестування, оскільки містить усю необхідну інформацію, що описує цей процес. Залежно від конкретизації описуваних завдань, тест-план може мати два рівні деталізації: майстер тест-план і детальний тест-план.

Детальний тест-план містить завдання тестування для кожної команди, для кожного релізу або ітерації проекту. Детальний тест-план створюється або для декомпозованих частин проекту, або для невеликих проектів і складається з:

– переліку областей тестування з пріоритетами;

– стратегії тестування;

– переліку можливих ризиків;

– переліку необхідних ресурсів;

– плану виконання проекту.

Майстер тест-план створюється або для організації процесу тестування між декількома командами, які тестують один проект, але мають різні завдання, або для проекту, який складається з безлічі ітерацій, які пов’язує якась загальна інформація, повторення якої в кожному релізі займає надто багато часу. Майстер тест-план містить:

– загальну інформацію про проект (посилання на документацію, баг-трекер, і т.д.);

– положення, що описують процес тестування, закладу дефектів і т. д.;

– критерії готовності продукту до випуску.

Існують кілька шаблонів тест-планів (IEEE, RUP).

**Баг репорт (bugreport)** – це технічний документ, який містить повний опис дефекту / бага з інформацією як про сам баг (короткий опис, серйозність, пріоритет і т. д.), так і про умови виникнення цього бага. Баг репорт повинен містити правильну, єдину термінологію, що описує елементи призначеного для користувача інтерфейсу та події цих елементів, що призводять до виникнення бага.

У загальному випадку, баг-репорт містить:

1) шапку:

– короткий опис (короткий опис проблеми);

– проект (назва поточного проекту);

– компонент додатка, у якому виник дефект;

– версія (версія збірки, у якому знайдено баг);

– серйозність (градація ступеня впливу на додаток бага);

– пріоритет (черга виправлення бага);

– статус (відображає статус бага в своєму життєвому циклі);

– автор (автор баг репорт);

– призначення (хто повинен виправити дефект);

2) оточення:

– операційна система, розрядність, браузер, його версія і т. д.;

3) опис:

– кроки відтворення (опис шляху, який призводить до виникнення дефекту);

– фактичний результат (результат, який отримується після виконання всіх кроків відтворення);

– очікуваний результат (результат, який бути відповідно до вимог);

4) додатки.

– приєднаний файл (логи, скріншоти, інші документи, які можуть допомогти відтворити проблему або розв’язати цю проблему).

Основні поля, присутність яких необхідна, у Додатку до баг репорті:

1) детальний опис багів:

– короткий опис. Поле, де розміщується призначення баг. Найчастіше, у короткому описі лаконічно відповідають на 3 питання: «Де?», «Що?», «Коли?» (саме в такій послідовності);

– серйозність. Дефект або повністю зупиняє працездатність програми, або тільки частину функціональності, або інше;

– кроки до відтворення. Точний і зрозумілий опис усіх кроків, які призводять до появи дефекту, з урахуванням усіх необхідних вхідних даних;

– фактичний результат;

– очікуваний результат.

– проект (назва поточного проекту);

– компонент додатка, у якому виник дефект;

– версія (версія збірки додатку, у якому знайдено баг);

– серйозність (градація ступеня впливу на додаток бага);

– пріоритет (черга виправлення бага);

– статус (відображає статус бага в своєму життєвому циклі);

– автор (автор баг репорт);

– призначення (хто повинен виправити дефект);

2) оточення:

– операційна система, розрядність, браузер, його версія і т. д.;

3) опис:

– кроки відтворення (опис шляху, який призводить до виникнення дефекту);

– фактичний результат (результат, який отримується після виконання всіх кроків відтворення);

– очікуваний результат (результат, який бути відповідно до вимог).

**Попередні зауваження до прикладів**

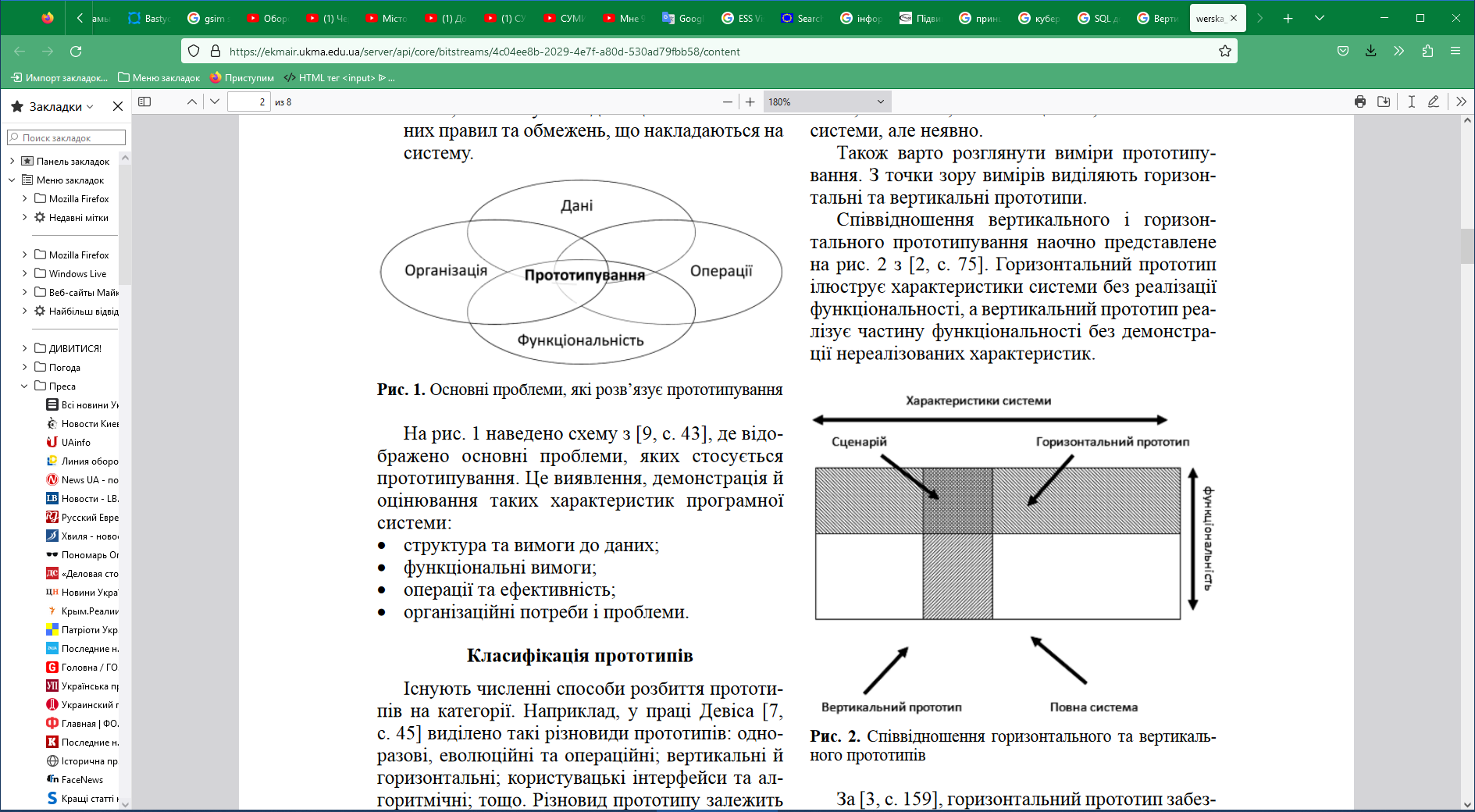


Рисунок -.1. Співвідношення горизонтального та вертикаль-ного прототипів

Горизонтальний прототип забезпечує широке представлення повної системи або підсистеми, зосереджуючись на користувацькому інтерфейсі більше, ніж на низькорівневій системній функціональності як, наприклад, доступ до бази даних. Горизонтальні прототипи використовують для підтвердження призначених для користувача інтерфейсних вимог і системного контексту, створення демонстраційних версій системи в маркетингових цілях, приблизної оцінки часових рамок і ресурсних затрат, яких потребуватиме проект.

Вертикальний прототип характерний повнішою розробкою деякої підсистеми або функції. Це корисно для отримання детальних вимог щодо певної функціональності. Вертикальне прототипування застосовують, наприклад, для вдосконаленого проектування бази даних, отримання інформації стосовно обсягів даних, системних інтерфейсних потреб, визначення характеристик мережі та вимог щодо потужностей, прояснення складних вимог, оскільки воно дає змогу побудувати весь ланцюг взаємодій фактичної системної функціональності.

**ПРИКЛАДИ**

Тест-план

| Вид вимоги | Роботи з тестування | Очікуваний результат |
| --- | --- | --- |
| **Функціональні вимоги** |  | |
| Вимоги користувача | Перевірка роботи системи, її можливості та правильність виконання | Згідно з вимогами: правильна робота всіх функцій |
| Простий та зрозумілий інтерфейс | Аналіз поведінки системи | Всі атрибути повинні мати варіанти вибору, наявність підказок, пояснень при неправильному введенні даних, простота формулювань текстової інформації інтерфейсів |
| Реалізувати можливість пошуку користувача за власним номером читача | Аналіз поведінки системи: тестується пошуковий інтерфейс на реалізацію пошуку при правильних та некоректно введених даних, можливість завдання частини номеру та отримання списку, які включають цю частину | Можливість пошуку користувача номером та частиною номера, отримання підказок у разі не правильного завдання номеру |
| Можливість створити власні записи(особистий кабінет читача) | Вертикальне прототипування, спрямоване на перевірку роботи особистого кабінету | Наявний кабінет читача, можливість пошуку в своєму кабінеті, збереження книг |
| Можливість працівникам додавати в базу даних дані про нові товари, покупців і читачів | Вертикальне прототипування, спрямоване на додавання нових записів | Працівники мають доступ до бази даних та можуть вносити нові дані, редагувати попередні |
| **Нефункціональні вимоги** |
| Доступ тільки зареєстрованих користувачів до архіву | При тестуванні під час спроби зайти на вкладку Архів отримуємо повідомлення про «Для доступу до архіву – спочатку зареєструйтеся» | Незареєстровані користувачі не мають доступ до архіву |
| Можливість завантаження і вивантаження файлів у форматі docx., pdf. | Прототипування горизонтальне, зокрема спроба завантаження файлів у форматі xlsx. або png не доступно | Користувач може завантажувати файли лише у форматі docx., pdf. |
| Реалізувати відображення підказок червоним кольором | Аналіз поведінки системи: при не правильно введених даних при пошуку, отримуємо підказку червоним кольором | Підказки мають бути червоним кольором |
| Надання коментарів та зауважень лише зареєстрованим користувачам | Спроба залишити коментар не зареєстрованим користувачем | Незареєстровані користувачі не можуть залишати коментарі |

Тест-кейс

| **№** | | **Завдання** | **Послідовність дій** | **Очікуваний результат** |
| --- | --- | --- | --- | --- |
| 1 | | Необхідно сформувати звіт всіх куплених книжок за останні 6 місяців | Звіт буде надано в середовищі Excel | відповідає вимогам |
| 2 | | Зміна шаблону чека покупки: обов’язково має бути вказана назва книги, її автор і літературний жанр | 1. Оформити замовлення на придбання книги 2. Зберегти замовлення 3. Натиснути на кнопку роздрукувати чек 4. Провірити правильність даних (чи вказана назва книги, автор і літературний жанр) | відповідає вимогам |
| 3 | | Треба змінити залежності , щоб покупець в залежності від обраного автора міг обрати назву книги, а не навпаки, оскільки назва книги можуть бути однакові у різних авторів. | 1. Використовуючи пошук Find author вказати ПІБ автора 2. Потім обрати потрібну книгу обраного раніше автора | відповідає вимогам |
| 4 | | Надати доступ молодшим працівникам додавати в базу даних дані про нові товари, покупців і читачів | 1. Відкрити вікно Library під користувачем : Продавець 2. Внести дані про книгу: автор, назва, рік 3. Натиснути Add book 4. Для перевірки внесених даних натиснути на Show Library | відповідає вимогам |
| 5 | | Для користувача Адміністратор(Бібліотекар) добавить можливість редагування назв, актуалізації жанрів, видалення книжок | 1. Зайти в систему під користувачем Адміністратор 2. Натиснути на Show Library 3. Обрати книгу для редагування 4. Внести виправлення 5. Зберегти 6. Провірити, натиснувши на Show Library | відповідає вимогам |
| 6 | | Реалізувати можливість пошуку користувача за власним номером читача | 1. Зайти в систему під користувачем Адміністратор 2. Натиснути на кнопку Знайти Користувача 3. Ввести номер 4. Натиснути на кнопку Show | відповідає вимогам |
| 7 | Можливість створити власні записи(особистий кабінет читача) | | 1. Зайти на сайт 2. Натиснути на Реєстрація користувача 3. Ввести логін та пароль 4. Натиснути на Зареєструватися | відповідає вимогам |