Тестування програмного забезпечення

Викладач: к.т.н. Шитікова Олена Вікторівна

Лекція 9
ТЕСТ-ДИЗАЙН ТА ТЕСТ-КЕЙСИ

Основні поняття та визначення тест-дизайну (test design)

Тест-дизайн (test design) – це етап процесу тестування програмного забезпечення, на якому проєктуються і створюються ***тестові випадки (тест-кейси)***, відповідно до визначених раніше критеріїв якості і цілей тестування.

Головні цілі тест-дизайну:

- написати тести, які виявлять найбільш серйозні помилки продукту;
- мінімізувати кількість тестів, необхідних для знаходження більшості серйозних помилок.

Основні поняття та визначення тест-дизайну (test design)

До завдань тест-дизайну відносять:

- написання тест-кейсів;
- аналіз ризиків;
- створення приймальних перевірок;
- опис процесу тестування;
- складання списку функцій продукту;
- аналіз вимог;
- розстановка пріоритетів тестування;
- аналіз скарг користувачів;
- побудова таблиць прийняття рішень.

Основні поняття та визначення тест-дизайну (test design)

План роботи над тест-дизайном:

- аналіз наявних проєктних артефактів: документація (специфікації, вимоги, плани), моделі, виконуваний код тощо;
- написання специфікації по тест-дизайну (test design specification);
- проєктування і створення **тестових випадків** (test cases).

Ролі, відповідальні за тест-дизайн:

- тест аналітик – визначає «ЩО тестувати?»;
- тест-дизайнер – визначає «ЯК тестувати?».

Основні поняття та визначення тест-дизайну (test design)

Навички, необхідні тест-дизайнеру:

- вміння поділяти систему на складові (робити декомпозицію);
- вміння збирати та аналізувати вимоги до продукту, якщо немає формально описаних вимог;
- вміння розставляти пріоритети;
- вміння формулювати свої думки (письмово та усно);
- знання і вміння застосовувати техніки тест-дизайну на практиці.

Техніки тест-дизайну

Техніки тест-дизайну – це методи створення тестів. Техніки містять, як теоретичну частину (деякі рекомендації по складанню тестів), а також практичну.

Найбільш поширені техніки тест-дизайну:

1. Еквівалентний поділ (Equivalence Partitioning)
2. Аналіз граничних значень (Boundary Value Analysis)
3. Причина/Наслідок (Cause/Effect)
4. Передбачення помилки (Error Guessing)
5. Вичерпне тестування (Exhaustive Testing)
6. Попарне тестування (Pairwise testing)

Найпоширеніші техніки тест-дизайну

- ***Еквівалентний поділ (Equivalence Partitioning)*** – це техніка, яка полягає в розбитті всього набору тестів на класи еквівалентності з подальшим скороченням кількості тестів.

Клас еквівалентності (Equivalence class) – це набір даних, що обробляється однаковим способом мислення й приводить до однакового результату.

Найпоширеніші техніки тест-дизайну

- ***Аналіз граничних значень (Boundary Value Analysis)*** – це техніка перевірки поведінки продукту на крайніх (граничних) значеннях вхідних даних.

Граничне тестування також може включати тести, що перевіряють поведінку системи на вхідних даних, що виходять за допустимий діапазон значень.

При цьому система повинна певним (заздалегідь обумовленим) способом обробляти такі ситуації.

Найпоширеніші техніки тест-дизайну

- ***Причина/наслідок (Cause/Effect)*** – це, як правило, введення комбінацій умов (причин), для отримання відповіді від системи (наслідків).

- ***Передбачення помилки (Error Guessing)*** – це коли тест-аналітик використовує свої знання системи та здатність до інтерпретації специфікації на предмет того, щоб «передбачити», за яких вхідних умов система може видати помилку.

Найпоширеніші техніки тест-дизайну

- ***Вичерпне тестування (Exhaustive Testing)*** – у межах цієї техніки перевіряються всі можливі комбінації вхідних значень, що має допомогти знайти всі проблеми.

На практиці застосування цього методу не є можливим через величезну кількість вхідних значень, застосовується лише у крайніх випадках.

Найпоширеніші техніки тест-дизайну

Вважається, що кожна техніка може складати ***рекомендації з семи питань:***

1. Що тестується?
2. Наскільки ретельно?
3. Ким тестується?
4. Які проблеми шукаються?
5. Як тестується?
6. Як оцінюється результат тестів?
7. Чи існує певна мета тестування?

Найпоширеніші техніки тест-дизайну

Завдання тест-аналітиків і дизайнерів зводиться до того, щоб, використовуючи різні стратегії і техніки тест-дизайну, створити набір тестових випадків, що забезпечує ***оптимальне тестове покриття*** (test coverage).

Однак, на більшості проєктів ці ролі не виділяються, а написання тест-кейсів виконують тестувальники.

Тестове покриття (Test Coverage)

Тестове покриття – це одна з метрик оцінки якості тестування, що являє собою щільність покриття тестами вимог або виконуваного коду.

Чим вище необхідний рівень тестового покриття, тим більше тестів буде вибрано для перевірки тестованих вимог або виконуваного коду.

Складність сучасного ПЗ та інфраструктури зробила неможливим проведення тестування з 100% тестовим покриттям. Тому для розробки набору тестів, що забезпечує більш-менш високий рівень покриття, можна використовувати *спеціальні інструменти* або *техніки тест-дизайну*.

Тестове покриття (Test Coverage)

Підходи до оцінки та виміру тестового покриття:

1. Покриття вимог (Requirements Coverage) – оцінка покриття тестами функціональних і нефункціональних вимог до продукту шляхом побудови матриць трасування (traceability matrix).

2. Покриття коду (Code Coverage) – оцінка покриття виконуваного коду тестами, шляхом відстеження неперевірених в процесі тестування частин ПЗ.

3. Тестове покриття на базі аналізу потоку управління – оцінка покриття, заснована на визначенні шляхів виконання коду програмного модуля та створення виконуваних тест-кейсів для покриття цих шляхів.

Тестове покриття (Test Coverage)

Метод покриття вимог зосереджений на перевірці відповідності набору проведених тестів вимогам до продукту, в той час як ***аналіз покриття коду*** – на повноті перевірки тестами розробленої частини продукту (вихідного коду), а ***аналіз потоку управління*** – на проходженні шляхів у графі або моделі виконання тестованих функцій (Control Flow Graph).

Метод оцінки покриття коду не виявить нереалізовані вимоги, оскільки він працює не з кінцевим продуктом, а з існуючим вихідним кодом.

Метод покриття вимог може залишити неперевіреними деякі ділянки коду, тому що не враховує кінцеву реалізацію.

Покриття вимог

Розрахунок тестового покриття щодо вимог проводиться за формулою:

$$T_{cov} = \left(\frac{L_{cov}}{L_{total}} \right) \cdot 100\%$$

де T_{cov} — тестове покриття;

L_{cov} — кількість вимог, що перевіряються тест-кейсами;

L_{total} — загальна кількість вимог.

Для вимірювання покриття вимог необхідно проаналізувати вимоги до продукту і розбити їх на пункти.

Опціонально кожен пункт зв’язується з тест-кейсами, які перевіряють його. Сукупність цих зв’язків і є матрицею трасування. Простеживши зв’язки, можна зрозуміти, які саме вимоги перевіряє тестовий випадок.

Покриття коду

Розрахунок тестового покриття щодо виконуваного коду програмного забезпечення проводиться за формулою:

$$T_{cov} = \left(\frac{L_{tc}}{L_{code}} \right) \cdot 100\%$$

де T_{cov} – тестове покриття;

L_{tc} – кількість рядків коду, покритих тестами;

L_{code} – загальна кількість рядків коду.

У даний час існує інструментарій, що дозволяє проаналізувати, які рядки були перевірені під час проведення тестування, завдяки чому можна значно збільшити покриття, додавши нові тести для конкретних випадків, а також позбутися від дублюючих тестів.

Тестове покриття на базі аналізу потоку управління коду

Тестування потоків управління (Control Flow Testing) – це одна з технік тестування білого ящика, заснована на визначенні шляхів виконання коду програмного модуля та створення виконуваних тест-кейсів для покриття цих шляхів.

Фундаментом для тестування потоків управління є побудова графів потоків управління (Control Flow Graph), основними блоками яких є:

- блок процесу – одна точка входу та одна точка виходу;
- точка альтернативи – одна точка входу, дві та більше точки виходу;
- точка з'єднання – дві та більше точок входу, одна точка виходу.

Тестове покриття на базі аналізу потоку управління коду

Рівень	*Назва*	*Короткий опис*
Рівень 0	-	«Тестуй все що зможеш протестувати, користувачі протестують інше». На англійській мові це звучить набагато елегантніше: *«Test whatever you test, users will test the rest».*
Рівень 1	Покриття операторів	Кожен оператор повинен бути виконаний як мінімум один раз.
Рівень 2	Покриття альтернатив. Покриття гілок	Кожен вузол з розгалуженням (альтернатива) виконаний як мінімум один раз.
Рівень 3	Покриття умов	Кожну умову, що має TRUE і FALSE на виході, виконано як мінімум один раз.
Рівень 4	Покриття умов альтернатив	Тестові випадки створюються для кожної умови і альтернативи.
Рівень 5	Покриття множинних умов	Досягається покриття альтернатив, умов та умов альтернатив (Рівні 2, 3 і 4).
Рівень 6	«Покриття нескінченного числа шляхів»	Якщо у разі зациклення, кількість шляхів стає нескінченним, допускається істотне їх скорочення, обмежуючи кількість циклів виконання для зменшення числа тестових випадків.
Рівень 7	Покриття шляхів	Всі шляхи повинні бути перевірені.

Тест-кейси (test cases)

Тестовий випадок або тест-кейс (test case) – це сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується, або її частини.

Тестовий набір (test suite) – набір тестів, що реалізують бізнес-завдання, виконуване тестованою системою.

Тестовий набір включає, окрім тестових сценаріїв, ще й тестові дані або правила їх генерації.

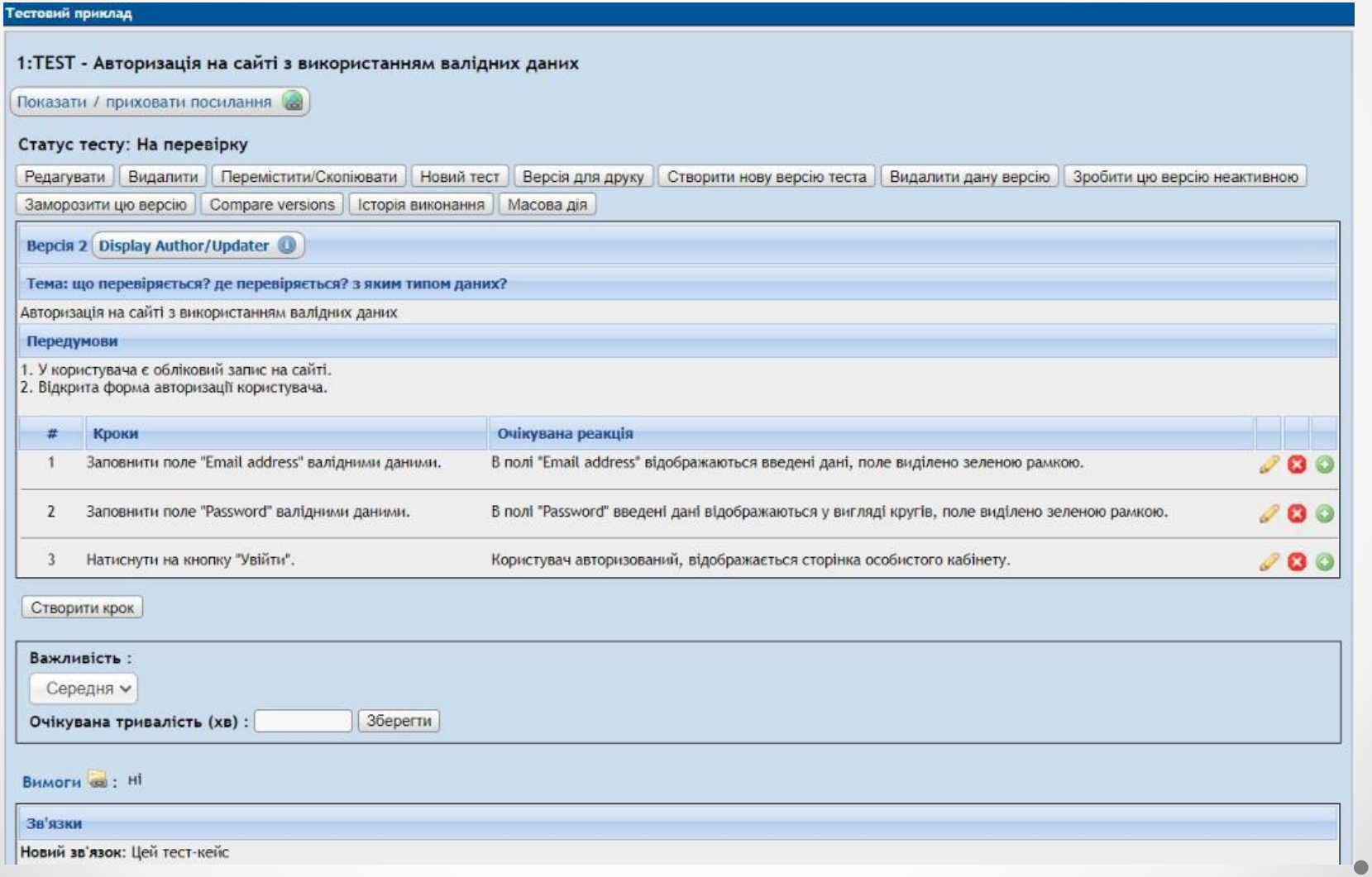
Тест-кейси (test cases)

До стандартних атрибутів тест-кейсу відносять:

- ID (Test Case ID);
- назва;
- тема (Summary);
- попередні умови (Preconditions);
- кроки для відтворення (Step actions);
- очікувані результати (Expected results);
- історія редагування;
- прикріплені файли (Attached files).

Тест-кейси (test cases)

Опис тест-кейсу в системі управління тестуванням TestLink



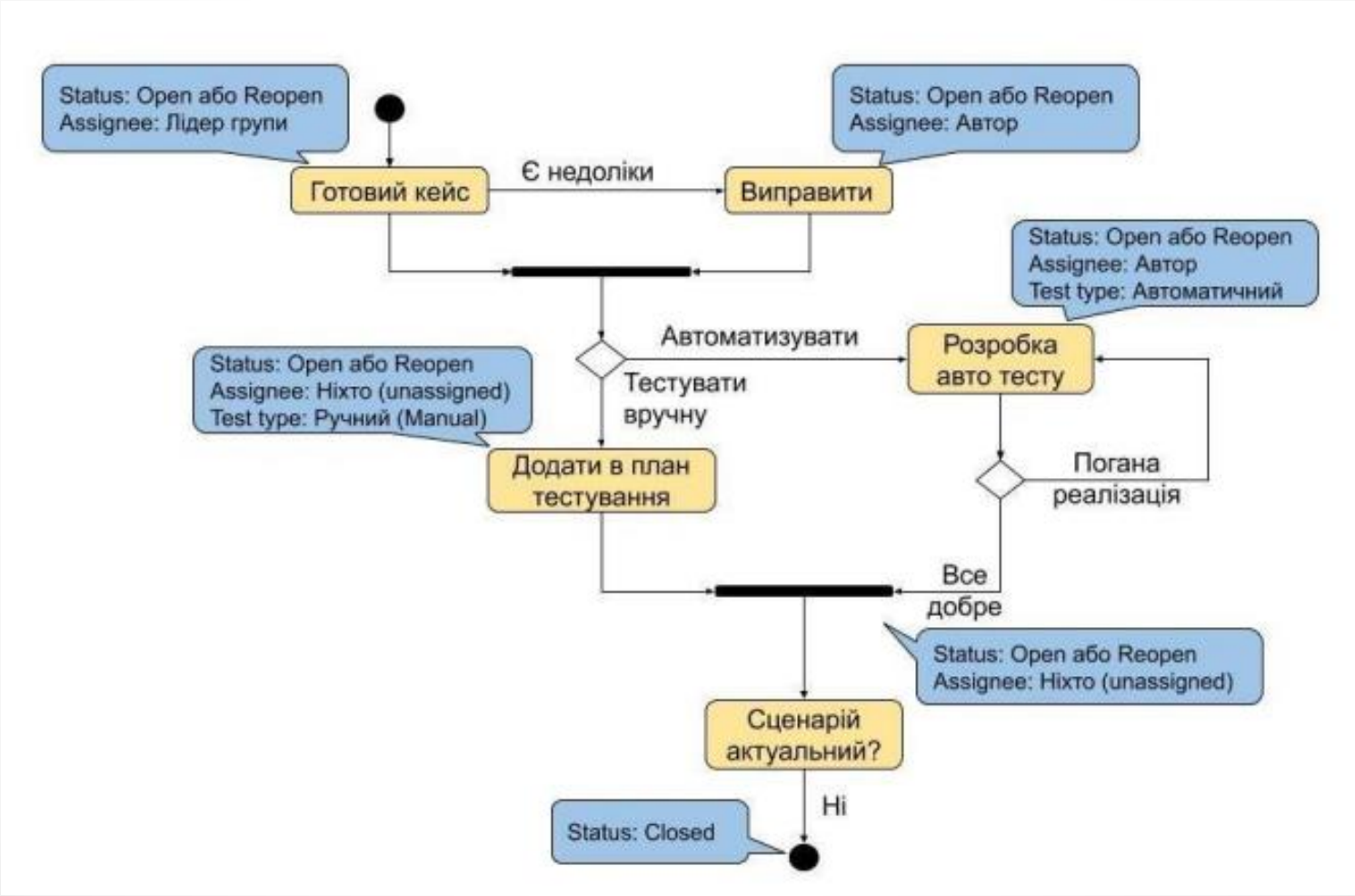
Тест-кейси (test cases)

Якісний тест-кейс повинен відповідати таким критеріям:

- тест-кейс повинен бути точно визначеним;
- тест-кейс повинен бути відтворюваним;
- набір тестів не повинен бути надмірним;
- тест повинен бути найкращим у своїй категорії;
- тест-кейс повинен бути коротким і простим;
- тест-кейс повинен бути однозначним і зрозумілим;
- тест-кейс не повинен бути занадто простим або занадто складним;
- іноді можна об’єднати два тести в один, але при цьому важливо не перестаратися.

Тест-кейси (test cases)

Життєвий цикл тестового випадку



Тест-кейси (test cases)

До переваг тестування за сценаріями відносять:

- Вимагає менше навичок від тестера, який виконує тести;
- Дає можливість легше оцінити тестове покриття;
- Стабільність та незмінність тест-кейсів дає можливість простежити якість продукту з часом;
- Забезпечує більш структурований підхід до тестування;
- Як правило, виконання тестів не вимагає додаткових досліджень та спілкування з розробниками, аналітиками, роботи зі специфікаціями, що дає виграш за часом.

Тест-кейси (test cases)

До недоліків тестування за сценаріями відносять:

- тестування з тест-кейсами обмежує свободу тестера;
- формує звички у тестуванні, тому що тестер звикає робити такі перевірки, як описані в тест-кейсах;
- тест-кейси потихеньку втрачають свою ефективність і перестають знаходити баги;
- постійне виконання одних і тих же тестів з часом стомлює;
- вимагає багато часу на підтримку тест-кейсів: оновлення, видалення, виправлення;
- тестер повільно розвиває своє знання продукту і контексту в порівнянні з дослідницьким тестуванням.

Тест-кейси (test cases)

До переваг дослідницького тестування відносять:

- добре розвиває навички, дає тестеру більше свободи;
- не вимагає часу для написання тестових сценаріїв, весь час приділяється тестуванню;
- заохочує вивчення тестером продукту.

Тест-кейси (test cases)

До недоліків дослідницького тестування відносять:

- недосвідченим тестерам в перший час буває важко через те, що у них немає прикладів тестів, які потрібно виконувати;
- без тест-кейсів складніше оцінити тестове покриття;
- вивчення продукту забирає час від безпосереднього тестування.

Тест-кейси (test cases)

При складанні тест-кейсів необхідно дотримуватися наступних принципів:

- тему тест-кейсу необхідно описувати за принципом «Що перевіряється? Де перевіряється? З яким типом даних (валідні, невалідні)?»;
- наявності опису тест-кейса;
- попередніх умов, де має бути вказано, яка форма, сторінка тощо відкрита;
- на кожен крок повинен бути очікуваний результат;

Тест-кейси (test cases)

– в кроках має бути зазначений тип даних, що вводяться (валідні/ невалідні), у разі невалідних – повинні бути наведені приклади таких даних;

– тест-кейс повинен бути завершеним і цілісним (наприклад, функція відновлення пароля повинна закінчуватися на успішній зміні пароля, а не на відправці листа з посиланням для зміни пароля).

Системи управління тестуванням (test management system)

Системи управління тестуванням використовуються для зберігання інформації про те, як належним чином проводити тестування, здійснення черговості проведення тестування відповідно до плану, а також для отримання інформації у вигляді звітів про стадії тестування та якість продукту, що тестується.

Інструменти мають різні підходи до тестування і, таким чином, включають різні набори функцій.

Системи управління тестуванням (test management system)

Зазвичай використовуються для планування ручного тестування, збору даних про результати проходження чеклістів та тест-кейсів, а також для отримання оперативної інформації у вигляді звітів.

Системи управління тестуванням допомагають оптимізувати процес тестування і забезпечують швидкий доступ до аналізу даних, засобів спільної роботи і більш якісну взаємодію між декількома проєктними групами. Багато систем управління тестуванням включають можливість роботи з вимогами.

Системи управління тестуванням (test management system)

Інструменти управління тестуванням дають командам можливість консолідувати і структурувати процес тестування за допомогою однієї з систем управління тестуванням замість установки декількох додатків, які призначені для управління тільки одним процесом або його частиною. Деякі програми включають передові інструментальні панелі для ретельного відстеження ключових показників, що дозволяє легко отримувати необхідну інформацію про стадії процесу тестування і якість продукту, що тестується.

Система управління тестуванням TestLink

TestLink – система управління тестуванням з вебінтерфейсом, яка дозволяє імпортувати вимоги, генерувати на їх основі тестові сценарії або створювати тестові сценарії з нуля, генерувати тестові специфікації, звіти про тестування, призначати виконання тестів на фахівців з контролю якості, відстежувати їх активність в реальному часі, пов’язувати не пройдені тести з баг-трекінговою системою і збирати різного роду статистику.

Система управління тестуванням TestLink

До переваг TestLink відносять:

- доменна аутентифікація, для якої досить встановити модуль php5-ldap і трохи підправити конфігурацію;
- оповіщення по e-mail про призначення завдань на тестування (коли призначаєте набори тестових випадків на спеціаліста з контролю якості, якому може автоматично приходити нотифікація на пошту);
- генерація тестової документації по шаблону, наближеному до корпоративного;
- інтеграція з Jira, що дає можливість пов’язувати внутрішньо системні повідомлення (ticket) в Jira з заблокованими або не пройденими тестовими випадками.

Система управління тестуванням TestRail

TestRail – система для управління даними, що отримані в результаті тестування, яка дозволяє вести тестову документацію та облік результатів виконання тестів. Даний інструмент допомагає відслідковувати процеси, керувати програмним забезпеченням і організовувати команду.

За допомогою ***TestRail*** можна створювати тестові випадки, управляти тестовими наборами та координувати весь процес тестування програмного забезпечення. ***TestRail*** надає можливість підвищити продуктивність та отримати повний огляд ходу процесу тестування.

Порівняльна характеристика систем управління тестуванням

Критерії порівняння	*TestLink*	*TestRail*
Створення проєкту	+	+
Ролі користувачів	+	+
Тест-план/тест-кейс	+	+
Вимоги	+	-
Прогон тестів	+	+
Звіти	+	+
Баг-трекер	-	-
Імпорт/Експорт	+	+
Інтеграція	+	+
SDK/звіти про креші	-	-
Автоматична збірка	-	-

Практичне застосування технік тест-дизайну при розробці тест-кейсів

Розробка тест-кейсів здійснюється за наступним планом:

1. Аналіз вимог.
2. Визначення набору тестових даних на основі технік еквівалетного поділу, аналізу граничних значень, передбачення помилки.
3. Розробка шаблону тесту на підставі техніки причина/наслідок.
4. Написання тест-кейсів на підставі первинних вимог, тестових даних та кроків тесту.

Практичне застосування технік тест-дизайну

Вимоги до функціональності форми прийому заявок

Елемент	*Тип елемента*	*Вимоги*
Тип звернення	combobox	Набір даних: 1. Консультація 2. Проведення тестування 3. Розміщення реклами 4. Помилка на сайті * На процес виконання операції прийому заявок не впливає.
Контактна особа	editbox	1. Обов'язкове для заповнення 2. Максимально 25 символів 3. Використання цифр та спец. символів не допускається
Контактний телефон	editbox	1. Обов'язкове для заповнення 2. Допустимі символи «+» та цифри 3. «+» можна використовувати тільки спочатку номера 4. Допустимі формати: – починається з плюса – 11-15 цифр; – +31612361264; – +375291438884; – без плюса – 5-10 цифр, наприклад: 0613261264, 2925167
Повідомлення	text area	1. Обов'язкове для заповнення 2. Максимальна довжина 1024 символів
Відправити	button	Стан: 1. За замовчуванням – не активна (Disabled) 2. Після заповнення обов'язкових полів стає активна (Enabled) Дії після натискання: 1. Якщо введені дані валідні – відправка повідомлення 2. Якщо введені дані невалідні – валідаційні повідомлення

Практичне застосування технік тест-дизайну

1. Аналіз вимог – це процес збору вимог до ПЗ, їх систематизації, документування, аналізу, виявлення суперечностей, неповноти, вирішення конфліктів у процесі розробки програмного забезпечення.

У процесі збору вимог важливо брати до уваги можливі протиріччя вимог різних зацікавлених осіб, таких як замовники та розробники.

Вимоги можуть бути функціональними та нефункціональними.

Практичне застосування технік тест-дизайну

Аналіз вимог включає три типи діяльності:

- **збір вимог**: спілкування з клієнтами та користувачами, щоб визначити, які їхні вимоги;
- **аналіз вимог**: визначення, чи є зібрані вимоги неясними, неповними, неоднозначними або суперечать одне одному; вирішення цих проблем;
- **документування вимог**: вимоги можуть бути задокументовані в різних формах, таких як простий опис, сценарії використання, призначені для користувача історії або специфікації процесів.

Практичне застосування технік тест-дизайну

2. Визначення набору тестових даних

На підставі вимог до полів та використанні техніки тест-дизайну визначається набір тестових даних:

• які поля необхідно перевірити на порожнє значення, так як можна викликати помилку;

• мінімальний набір даних;

• поля із комбінованим типом (цифри використовуються спільно з символами);

• можна додати деяку кількість значень на підставі особистого досвіду (техніка EG).

Практичне застосування технік тест-дизайну

3. Вибір тестових даних для поля «Контактна особа»

Це обов'язкове поле розміром від 1 до 25 символів (включаючи граничні значення). Перевірка на обов'язковість додає до тестових даних порожнє значення.

Проведемо аналіз граничних умов (BVA), отримаємо набір: 0, 1, 2, 24, 25 і 26 символів.

У зв'язку з тим, що значення 2 і 24 символу є некритичними, їх можна не додавати.

У підсумку отримуємо мінімальний набір даних для тестування поля – це рядки 1 і 25 – ОК, і 0 (пусте значення), 26 символів – NOK.

Практичне застосування технік тест-дизайну

Дані для подальшого використання при складанні тест-кейсів

Поле	*OK/ NOK*	*Значення*	*Коментар*
Тип звернення	OK	Консультація	перший у списку
		Помилка на сайті	останній у списку
	NOK		
Контактна особа	OK	йцукенгшщзйцукенгшщзйцуке	25 символів нижній регістр
		а	1 символ
		ЙЦУКЕНГШЩЗФИВАПРОЛДЖЯЧСМІ	25 символів верхнього регістру
		ИЦУКЕНГШЩЗфивапролджЯЧСМІ	25 символів змішаний регістр
	NOK		порожнє значення
		йцукенгшщзйцукенгшщзйцукей	довжина більше максимальної (26 символів)
		@ # $% ^ &;.?,> | \ / № «! () _ {} [<~	спец.символи (ASCII)
		+12345678901234567890123456789012345	тільки цифри
		adsadasdasdas dasdasd asasdsads (...) sas	дуже довгий рядок (~ 1Mb)
Контактний телефон	OK	+12345678901	з плюсом - мінімальна довжина
		+123456789012345	з плюсом – максимальна довжина
		1 2 3 4 5	без плюса – мінімальна довжина
		1234567890	без плюса – максимальна довжина
	NOK		порожнє значення
		+1234567890	з плюсом – <мінімальної довжини
		+1234567890123456	з плюсом –> максимальної довжини
		1234	без плюса – <мінімальної довжини
		12345678901	без плюса -> максимальної довжини
		+ YYYXXXyyyxxzz	з плюсом –букви замість цифр
		yyyxxxxzz	без плюса – букви замість цифр
		+ ### - $$$ -% ^ - & ^ - &!	спец.символи (ASCII)
		+12323123231232132312 32 (...) 99	дуже довгий рядок (~ 1Mb)
Повідомлення	OK	йццуйцуйц (...) йцу	максимальна довжина (1024 символу)
	NOK		порожнє значення
		йццуйцуйц (...) йцуц	довжина більша максимальної (1025 символів)
		adsadasdasdas dasdasd asasdsads (...) sas	дуже довгий рядок (~ 1Mb)
		@ ## $$$% ^ & ^ &	тільки спец.символи (ASCII)

Практичне застосування технік тест-дизайну

4 Розроблення шаблону тесту

Дія	*Очікуваний результат*
1. Відкрити форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка «Відправити» не активна
2. Заповнити поля форми: Тип звернення Контактна особа Контактний телефон Повідомлення	Поля заповнені Кнопка «Відправити» – активна (Enabled)
3. Натиснути кнопку «Відправити»	Якщо введені дані коректні: Повідомлення «Заявку відправлено» виведено на екран. Нова заявка з'явилася в списку на сторінці «Заявки». Якщо введені дані НЕ коректні: Валідаційні сполучення з усіма помилками виведено на екран. Заявка НЕ з'явилася в списку на сторінці «Заявки».

Практичне застосування технік тест-дизайну

5. Написання тест-кейсів на підставі первинних вимог, тестових даних і шаблону тесту

Послідовний перебір – це перебір всіх можливих комбінацій наявних значень. Таким чином, виходить, що кількість тест-кейсів буде дорівнювати добутку кількості варіантів тестових даних для кожного поля.

Попарний перебір (Pairwise Testing) дозволяє створити тестові набори, які комбінують дані з двох полів. Завдяки цьому, кількість отриманих на виході тест-кейсів у рази менша, ніж при комбінуванні того ж набору даних при послідовному переборі. Алгоритми генерації комбінацій для попарного тестування: Orthogonal Arrays Testing, All pairs, IPO (In-Parameter Order).

Практичне застосування технік тест-дизайну

Позитивний тест-кейс (всі поля ОК)

Дія	*Очікуваний результат*
1. Відкрити форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка «Відправити» не активна
2. Заповнити поля форми: Тип звернення= Консультація Контактна особа = йцукенгшщзйцукенгшщзйцуке Контактний телефон = + 7-916-111-11-11 Повідомлення	Поля заповнені Кнопка «Відправити» – активна (Enabled)
3. Натиснути кнопку «Відправити»	Повідомлення «Заявку відправлено» виведено на екран. Нова заявка з'явилася в списку на сторінці «Заявки».

Практичне застосування технік тест-дизайну

Негативний тест-кейс (поле «Контактна особа» NOK)

Дія	*Очікуваний результат*
1. Відкрити форму відправки повідомлення	Форма відкрита Всі поля за замовчуванням порожні Обов'язкові поля позначені – * Кнопка «Відправити» не активна
2. Заповнити поля форми: Тип звернення = Консультація Контактна особа = @ # $% ^ &;.?,> | \ / № «! () _ {} [<~ Контактний телефон = (916) 333-33-33 Повідомлення = йццуйцуйц (...) йцу - 1024 символів	Поля заповнені Кнопка «Відправити» – активна (Enabled)
3. Натиснути кнопку «Відправити»	Валідаційні сполучення з усіма помилками виведено на екран: У полі «Контактна особа» заборонено використання цифр та спец. символів. Заявка не з'явилась в списку на сторінці «Заявки».