Лекція 3
Чек-листи, тест-кейси, набори тест-кейсів

3.1. Чек лист

Як легко можна зрозуміти з попередніх розділів, тестувальнику доводиться працювати з величезною кількістю інформації, вибирати з безлічі варіантів вирішення завдань та винаходити нові. У процесі цієї діяльності об'єктивно неможливо пам'ятати усі думки, а тому продумування та розробку тест-кейсів рекомендується виконувати з використанням «чек-листів».

Чек-лист - набір ідей. Чек-лист — це просто набір ідей: ідей із тестування, ідей із розробки, ідей із планування та управління — будь-яких ідей.

Чек-лист найчастіше є звичайним і звичний нам список:

у якому послідовність пунктів немає значення (наприклад, список значень якогось поля);

в якому послідовність пунктів важлива (наприклад, кроки в стислої інструкції);

структурований (багаторівневий) список, що дозволяє відобразити ієрархію ідей.

Важливо зрозуміти, що немає і не може бути жодних заборон та обмежень при розробці чек-листів — головне, щоб вони допомагали у роботі. Іноді чек-листи можуть навіть висловлюватися графічно (наприклад, з використанням ментальних або концепт-карт), хоча традиційно їх складають у вигляді багаторівневих списків.

Оскільки в різних проектах зустрічаються однотипні завдання, добре продумані та акуратно оформлені чек-листи можуть використовуватися повторно, чим досягається економія сил та часу.

Дуже частим є питання, чи потрібно в чек-листах писати очікувані результати. У класичному розумінні чек-листа – ні (хоч це не заборонено), так як чек-лист – це набір ідей, деталізація яких у вигляді кроків і очікуваних результатів буде у тест-кейсах. Але очікувані результати можуть додаватися, наприклад, у таких випадках:

- у певному пункті чек-листа розглядається особлива, нетривіальна поведінка програми чи складна перевірка, результат якої важливо відзначити вже зараз, щоб не забути;

- в силу стислих термінів або нестачі інших ресурсів тестування проводиться безпосередньо за чек-листами без тест-кейсів.

Для того щоб чек-лист був дійсно корисним інструментом, він повинен мати ряд важливих властивостей.

Логічність. Чек -лист пишеться не «просто так», а на основі цілей і для того, щоб допомогти у досягненні цих цілей. На жаль, однією з найчастіших і найнебезпечніших помилок при складанні чек-листа є перетворення його на звалище думок, які ніяк не пов'язані один з одним.

Послідовність та структурованість. Зі структурованістю все досить просто - вона досягається за рахунок оформлення чек-листа у вигляді багаторівневого списку. Щодо послідовності, то навіть у тому випадку, коли пункти чек-листа не описують ланцюжок дій, людині все одно зручніше сприймати інформацію у вигляді деяких невеликих груп ідей, перехід між якими є зрозумілим і очевидним (наприклад, спочатку можна прописати ідеї простих позитивних тестів-кейсів, потім ідеї простих негативних тест-кейсів, потім поступово підвищувати складність тест-кейсів, але не варто писати ці ідеї упереміш).

Повнота і надмірність . Чек-лист повинен бути акуратною «сухою вичавкою» ідей, у яких немає дублювання (які часто з'являються через різні формулювання однієї й тієї ж ідеї), і в той же час ніщо важливе не втрачено.

Правильно створювати та оформлювати чек-листи також допомагає сприйняття їх не лише як сховища наборів ідей, а й як «вимоги для складання тест-кейсів». Ця думка призводить до перегляду та переосмислення властивостей якісних вимог щодо застосування до чек-листів.

Отже, розглянемо процес створення чек-листа.

Оскільки ми не можемо відразу «протестувати весь застосунок» (це занадто велике завдання, щоб вирішити її одним махом), нам уже зараз потрібно вибрати якусь логіку побудови чек-листів — їх буде кілька (в результаті їх можна буде структуровано об'єднати в один, але це обов'язково).

Типовими варіантами такої логіки є створення окремих чек-листів для:

типових користувальницьких сценаріїв;

різних рівнів функціонального тестування;

окремих частин (модулів та підмодулів) застосунка;

окремих вимог, груп вимог, рівнів та типів вимог;

частин або функцій програми, найбільш схильних до ризиків.

Цей список можна розширювати та доповнювати, можна комбінувати його пункти, отримуючи, наприклад, чек-листи для перевірки найбільш типових сценаріїв, які стосуються певної частини програми.

Щоб проілюструвати принципи побудови чек-листів, ми скористаємося логікою розбиття функцій програми за ступенем їхньої важливості на три категорії:

1. Базові функції, без яких існування програми втрачає сенс (тобто найважливіші — те, заради чого застосунок взагалі створювалося), або порушення роботи яких створює об'єктивні серйозні проблеми для середовища виконання.
2. Функції, затребувані більшістю користувачів у їхній повсякденній роботі.
3. Інші функції (різноманітні «дрібниці», проблеми з якими не сильно вплинуть на цінність програми для кінцевого користувача).

Функції, без яких існування програми втрачає сенс.

Спочатку наведемо весь чек-лист для димового тестування, а потім розберемо його докладніше.

- Конфігурування та запуск.
- Обробка файлів:

		Формати вхідних файлів		
		TXT	HTML	MD
Кодування вхідних файлів	WIN1251	+	+	+
	CP866	+	+	+
	KOI8R	+	+	+

- Зупинка

Тут перелічені всі ключові функції програми.

Конфігурування та запуск. Якщо програму неможливо налаштувати для роботи в середовищі користувача, вона марна. Якщо програма не запускається, вона марна. Якщо на стадії запуску виникають проблеми, вони можуть негативно позначитися на функціонуванні програми і тому також заслуговують на пильну увагу.

Обробка файлів. Заради цього застосунок і розроблявся, тому тут навіть на стадії створення чек-листа ми не полінувалися створити матрицю, яка відображатиме всі можливі комбінації допустимих форматів та допустимих кодувань вхідних файлів, щоб нічого не забути та підкреслити важливість відповідних перевірок.

Зупинка. З точки зору користувача ця функція може не здаватися настільки вже важливою, але зупинка (і запуск) будь-якої програми пов'язані з великою кількістю системних операцій, проблеми з якими можуть призвести до багатьох серйозних наслідків (аж до неможливості повторного запуску програми або порушення роботи операційної системи).

Функції, затребувані більшістю користувачів

Наступним кроком ми будемо виконувати перевірку того, як застосунок поводиться у звичайному повсякденному житті, поки не торкаючись екзотичних ситуацій. Дуже частим питанням є допустимість дублювання перевірок на різних рівнях функціонального тестування — чи можна так робити. Одночасно і "ні", і "так". «Ні» у тому сенсі, що не допускається (не має сенсу) повторення тих самих перевірок, які щойно були виконані. «Так» у тому сенсі, що будь-яку перевірку можна деталізувати та забезпечити додатковими деталями.

- *Конфігурування та запуск* :
    - з правильними параметрами;
    - без параметрів;
    - із недостатньою кількістю параметрів;
    - з неправильними параметрами.
- *Обробка файлів* :
    - Різні формати, кодування та розміри:
    - Недоступні вхідні файли: немає доступу; файл відкритий та заблокований; файл із атрибутом "тільки для читання".
- *Зупинка* :
    - Закриття вікна консолі.
- *Журнал роботи програми* :
    - Автоматичне створення (за відсутності журналу).
    - Продовження (доповнення журналу) під час повторних запусків.

- *Продуктивність* :
    - Елементарний тест із грубою оцінкою.

Зверніть увагу, що чек-лист може містити не тільки «гранично стислі тези», а й цілком розгорнуті коментарі, якщо це необхідно – краще пояснити суть ідеї докладніше, ніж потім гадати, що було на увазі.

Також зверніть увагу, що багато пунктів чек-листа мають дуже високорівневий характер, і це нормально. Наприклад, «пошкодження в допустимому форматі» звучить розпливчасто, але цей недолік буде усунено вже на рівні повноцінних тест-кейсів.

Інші функції та особливі сценарії

Настав час звернути увагу на різноманітні дрібниці та хитрі нюанси, проблеми з якими навряд чи сильно стурбують користувача, але формально все ж таки вважатимуть помилками.

- *Конфігурування та запуск* :
    - Значення SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME у різних стилях різні;
    - Розмір LOG_FILE_NAME на момент запуску: 2-4 ГБ. або 4+ ГБ.
    - Запуск двох та більше копій програми з однаковими параметрами SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME;
- *Обробка файлів*: файл вірного формату, в якому текст представлений у двох і більше кодуваннях одночасно.

Як ми побачимо далі, створення якісного тест-кейсу може вимагати тривалої копіткої і досить монотонної роботи, яка при цьому не вимагає від досвідченого тестувальника сильних інтелектуальних зусиль, а тому перемикання між роботою над чек-листами (творча складова) і розписуванням їх у тест-кейси (механічна складова) дозволяє урізноманітнити робочий процес та знизити стомлюваність. Хоча, звичайно, написання складних і якісних тест-кейсів може виявитися анітрохи не менш творчою роботою, ніж продумування чек-листів.

3.2. Тест-кейс та його життєвий цикл

3.2.1. Термінологія. Для початку визначимося з термінологією, оскільки тут є багато плутанини, викликаної різними перекладами англомовних термінів та різними традиціями у тих чи інших країнах, фірмах та окремих командах.

На чолі лежить термін «тест». Офіційне визначення звучить так.

Тест - набір з одного або кількох тест-кейсів.

Оскільки серед усіх інших термінів цей найлегше і найшвидше вимовляти, залежно від контексту під ним можуть розуміти і окремий пункт чек-листа, і окремий крок у тест-кейсі, і сам тест-кейс, і набір тест-кейсів і так далі. Головне тут одне: якщо ви чуєте чи бачите слово тест, сприймайте його в контексті.

Тепер розглянемо найголовніший для нас термін – «тест-кейс».

Тест-кейс - набір вхідних даних, умов виконання та очікуваних результатів, розроблений з метою перевірки тієї чи іншої властивості або поведінки програмного засобу.

Під тест-кейсом також може розумітися відповідний документ, який представляє формальний запис тест-кейсу. Якщо у тест-кейсу не вказані вхідні дані, умови виконання та очікувані результати, або не ясна мета тест-кейсу — це поганий тест-кейс (іноді він не має сенсу, іноді його взагалі неможливо виконати).

Інші терміни, пов'язані з тестами, тест-кейсами та тестовими сценаріями, на даному етапі можна прочитати просто з ознайомлювальною метою. Якщо ви відкриєте ISTQB-глосарій на літеру «Т», ви побачите величезну кількість термінів, тісно пов'язаних один з одним перехресними посиланнями: на початковому етапі вивчення тестування немає необхідності глибоко розглядати їх усі, проте деякі все ж таки заслуговують на увагу. Вони представлені нижче.

Високорівневий тест-кейс - тест-кейс без конкретних вхідних даних та очікуваних результатів. Як правило, обмежується загальними ідеями та операціями, схожим за своєю суттю з докладно описаним пунктом чек-листа. Досить часто зустрічається в інтеграційному тестуванні та системному тестуванні, а також на рівні димового тестування. Може бути відправною точкою для проведення дослідницького тестування або для створення низькорівневих тест-кейсів.

Низькорівневий тест-кейс — тест-кейс із конкретними вхідними даними та очікуваними результатами. Являє собою цілком готовий до виконання тест-кейс і взагалі є найбільш класичним видом тест-кейсів. Тестувальників-початківців найчастіше вчать писати саме такі тести, так як прописати всі дані докладно - набагато простіше, ніж зрозуміти, яку інформацію можна знехтувати, при цьому не знизивши цінність тест-кейсу.

Специфікація тест-кейсу - документ, що описує набір тест-кейсів (включаючи їх цілі, вхідні дані, умови та кроки виконання, очікувані результати) для елемента, що тестується.

Специфікація тесту – документ, що складається зі специфікації тест-дизайну, специфікації тест-кейсу або специфікації тест-процедури.

Тест-сценарій - документ, що описує послідовність дій щодо виконання тесту (також відомий як "тест-скрипт").

3.2.2. Мета написання тест-кейсів. Тестування можна проводити і без тест-кейсів (не потрібно, але можна; ефективність такого підходу варіюється в дуже широкому діапазоні в залежності від багатьох факторів). Наявність тест-кейсів дозволяє:

- Структурувати та систематизувати підхід до тестування (без чого великий проект майже гарантовано приречений на провал).
- Обчислювати метрики тестового покриття та вживати заходів щодо його збільшення (тест-кейси тут є головним джерелом інформації, без якого існування подібних метрик втрачає сенс).
- Відслідковувати відповідність поточної ситуації плану (скільки приблизно знадобиться тест-кейсів, скільки вже є, скільки виконано із запланованої на даному етапі кількості тощо).
- Уточнити взаєморозуміння між замовником, розробниками та тестувальниками (тест-кейси часто набагато наочно показують поведінку застосунка, ніж це відображено у вимогах).
- Зберігати інформацію для тривалого використання та обміну досвідом між співробітниками та командами (або як мінімум – не намагатися втримати в голові сотні сторінок тексту).
- Проводити регресійне тестування та повторне тестування (які без тест-кейсів було б взагалі неможливо виконати).
- Підвищувати якість вимог (ми це вже розглядали: написання чек-листів та тест-кейсів – хороша техніка тестування вимог).

- Швидко вводити в курс справи нового співробітника, який нещодавно підключився до проекту.

3.2.3. Життєвий цикл тест-кейс. На відміну від звіту про дефект, у якого є повноцінний розвинений життєвий цикл, для тест-кейсу йдеться швидше про набір станів (рис. 3.1), в яких він може знаходитися (жирним шрифтом відзначені найбільш важливі стани).

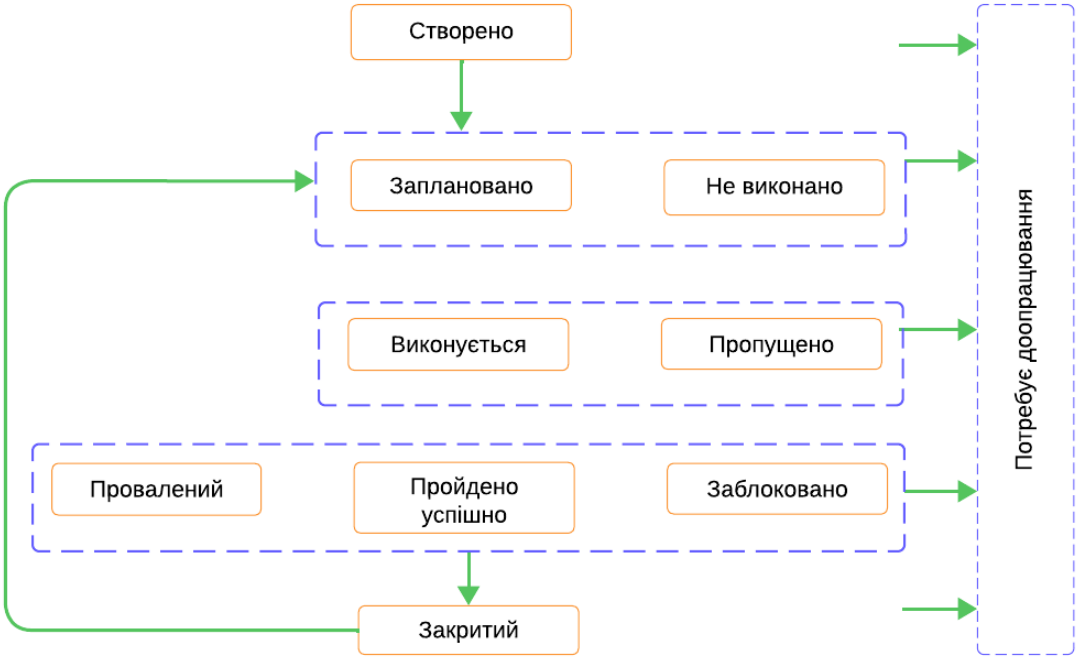

Рисунок 3.1 - Життєвий цикл тест-кейсу

Створено - типовий початковий стан практично будь-якого артефакту. Тест-кейс автоматично перетворюється на цей стан після створення.

Заплановано — у цьому стані тест-кейс знаходиться, коли він або явно включений до плану найближчої ітерації тестування, або, як мінімум, готовий до виконання.

Не виконано — у деяких системах управління тест-кейсами цей стан замінює собою попередній (запланований). Знаходження тест-кейсу в даному стані означає, що він готовий до виконання, але не був виконаний.

Виконується — якщо тест-кейс вимагає тривалого часу виконання, він може бути переведений у цей стан для підкреслення того факту, що робота йде, і незабаром очікується її результатів. Якщо виконання тест-кейсу займає мало часу, цей стан, як правило,

пропускається, а тест-кейс одразу переводиться в один із трьох наступних станів — «провалений», «пройдено успішно» або «заблоковано».

Пропущено - бувають ситуації, коли виконання тест-кейсу скасовується з міркувань нестачі часу або зміни логіки тестування .

Провален — цей стан означає, що в процесі виконання тест-кейсу було виявлено дефект, який полягає в тому, що очікуваний результат принаймні за один крок тест-кейсу не збігається з фактичним результатом. Якщо в процесі виконання тест-кейсу був «випадково» виявлений дефект, ніяк не пов'язаний з кроками тест-кейсу та їх очікуваними результатами, тест-кейс вважається успішним пройденим (при цьому, природно, за виявленим дефектом створюється звіт про дефект).

Пройдено успішно — цей стан означає, що в процесі виконання тест-кейсу не було виявлено дефектів, пов'язаних із розходженням очікуваних та фактичних результатів його кроків.

Заблокований - цей стан означає, що з якоїсь причини виконання тест-кейсу неможливе (як правило, такою причиною є наявність дефекту, що не дозволяє реалізувати якийсь сценарій користувача).

Закритий дуже рідкісний випадок, так як тест-кейс, як правило, залишають у станах «провалений/ пройдений успішно/ заблокований/ пропущений». У цей стан у деяких системах управління тест-кейс переводять, щоб підкреслити той факт, що на даній ітерації тестування всі дії з ним завершені.

Вимагає доопрацювання — як видно зі схеми, в цей стан (і з нього) тест-кейс може бути переведений у будь-який момент часу, якщо в ньому буде виявлено помилку, якщо зміняться вимоги, за якими він був написаний, чи настане інша ситуація, не що дозволяє вважати тест-кейс придатним для виконання та переведення в інші стани.

Ще раз підкреслимо, що на відміну від життєвого циклу дефекту, який досить стандартизований та формалізований, для тест-кейсу описане вище носить загальний рекомендаційний характер, розглядається скоріше як розрізнений набір станів (а не суворий життєвий цикл) і може відрізнятися в різних компаніях (у зв'язку з наявними традиціями чи можливостями систем управління тест-кейсами).

3.3. Атрибути (поля) тест-кейсу

Як було зазначено вище, термін «тест-кейс» може ставитися до формальної запису тест-кейса як технічного документа. Цей запис має загальноприйняту структуру, компоненти якої називаються атрибутами (полями) тест-кейсу.

Залежно від інструменту керування тест-кейсами зовнішній вигляд їх запису може трохи відрізнятися, можуть бути додані або прибрані окремі поля, але концепція залишається незмінною.

Загальний вигляд усієї структури тест-кейсу представлений на рис. 3.2.

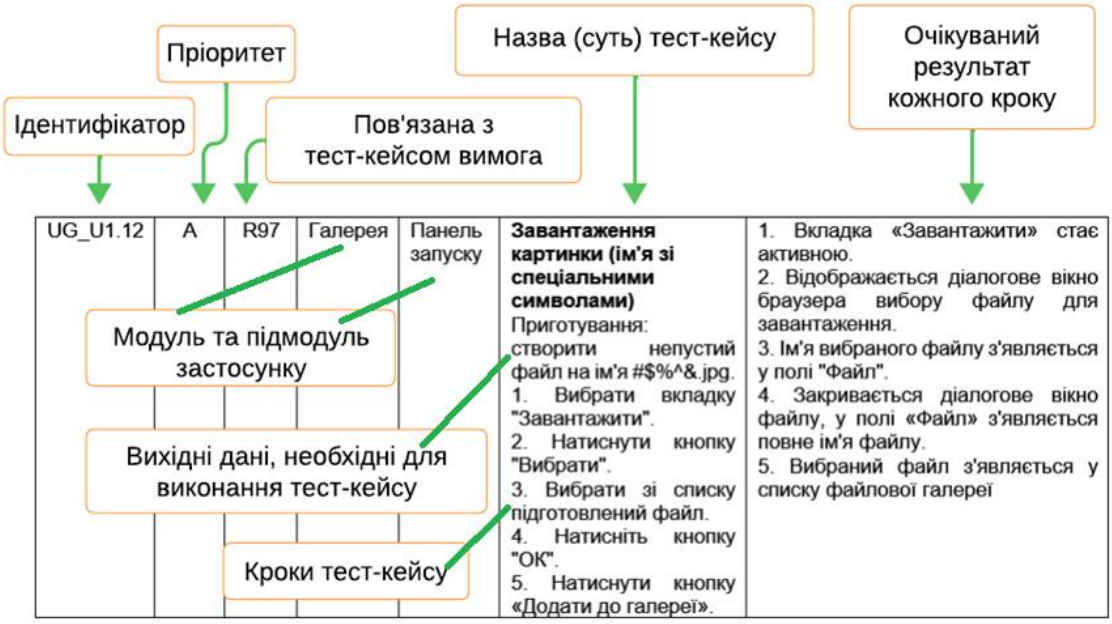

Рисунок 3.2 - Загальний вигляд тест-кейсу

Розглянемо кожний атрибут докладно.

Ідентифікатор є унікальним значенням, що дозволяє однозначно відрізнити один тест-кейс від іншого і використовується в різних посиланнях. У загальному випадку ідентифікатор тест-кейсу може бути просто унікальним номером, але (якщо дозволяє інструментальний засіб управління тест-кейсами) може бути і куди складніше: включати префікси, суфікси та інші осмислені компоненти, що дозволяють швидко визначити мету тест-кейсу та частину застосунку, до якого він належить.

Пріоритет показує важливість тест-кейсу. Він може бути виражений буквами (A, B, C, D, E), цифрами (1, 2, 3, 4, 5), словами («вкрай високий», «високий», «середній», «низький», «вкрай низький»)

або іншим зручним способом. Кількість градацій також не фіксована, але найчастіше лежить у діапазоні від трьох до п'яти.

Пріоритет тест-кейсу може корелювати з:

- важливістю вимоги, сценарію користувача або функції, з якими пов'язаний тест-кейс;
- потенційною важливістю дефекту, на пошук якого спрямований тест-кейс;
- ступенем ризику, пов'язаного з тест-кейсом, що перевіряється вимогою, сценарієм або функцією.

Основне завдання цього атрибуту — спрощення розподілу уваги та зусиль команди (вищі пріоритетні тест-кейси отримують їх більше), а також спрощення планування та прийняття рішення про те, чим можна пожертвувати в якійсь форс-мажорній ситуації, що не дозволяє виконати всі заплановані тест-кейси.

Пов'язана з тест-кейсом вимога показує ту основну вимогу, перевірці якої присвячено тест-кейс (основне - тому, що один тест-кейс може торкатися кількох вимог). Наявність цього поля покращує таку властивість тест-кейсу, як простежуваність.

Модуль і підмодуль програми вказують на частини програми, до яких належить тест-кейс, і дозволяють краще зрозуміти його мету. Ідея поділу програми на модулі і підмодулі випливає з того, що в складних системах практично неможливо охопити поглядом весь проект цілком, і питання «як протестувати цю програму» стає неприпустимо складним. Тоді застосунок логічно поділяється на компоненти (модулі), а ті, у свою чергу, на дрібніші компоненти (підмодулі). І ось вже для таких невеликих частин застосунка придумати чек-листи та створити хороші тест-кейси стає набагато простіше.

Як правило, ієрархія модулів і підмодулів створюється як єдиний набір для всієї проектної команди, щоб унеможливити плутанину через те, що різні люди будуть використовувати різні підходи до такого поділу або навіть просто різні назви одних і тих же частин програми.

Але що робити, якщо ми не знаємо «нутрощів» програми (або не дуже розуміємось на програмуванні)? Модулі та підмодулі можна виділяти на основі графічного інтерфейсу користувача (великі області та елементи всередині них), на основі розв'язуваних застосунком задач та підзавдань і т.д. Головне, щоб ця логіка була однаково застосована до всього застосунку.

Слід запам'ятати, що модуль та підмодуль застосунка — це НЕ дії, це саме структурні частини, «шматки» застосунка. В оману вас можуть ввести такі назви, як, наприклад, «друк, налаштування принтера» (але тут маються на увазі саме частини програми, що відповідають за друк та за налаштування принтера, а не процес друку або налаштування принтера).

Наявність полів «Модуль» та «Підмодуль» покращує таку властивість тест-кейсу, як простежуваність.

Назва (сутність) тест-кейсу покликане спростити і прискорити розуміння основної мети тест-кейсу без звернення до його інших атрибутів. Саме це поле є найбільш інформативним під час перегляду списку тест-кейсів.

Назва тест-кейсу може бути повноцінною пропозицією, фразою, набором словосполучень - головне, щоб виконувались такі умови:

- Інформативність.
- Хоча б відносна унікальність (щоб не плутати різні тест-кейси).

Якщо інструмент управління тест-кейсами не вимагає писати назву, його все одно треба писати. Тест-кейси без назв перетворюються на суміш інформації, використання якої пов'язане з колосальними і абсолютно безглуздими витратами.

І ще одна маленька ідея, яка допоможе краще формулювати назви. У дослівному перекладі з англійської «test case» означає «тестовий випадок». Так ось, назва таки визначає цей випадок, тобто що відбувається у тест-кейсі, яку ситуацію він перевіряє.

Вихідні дані, необхідні для виконання тест-кейсу, дозволяють описати все те, що має бути підготовлено до початку виконання тест-кейсу, наприклад:

- Стан бази даних.
- Стан файлової системи та її об'єктів.
- Стан серверів та мережної інфраструктури.

Те, що описується в цьому полі, готується без використання програми, що тестується, і таким чином, якщо тут виникають проблеми, не можна писати звіт про дефект у застосунку. Ця думка дуже важлива, тому пояснимо її простим життєвим прикладом. Уявіть, що ви дегустуєте цукерки. У полі «вихідні дані» можна прописати «купити цукерки таких сортів у такій кількості». Якщо таких цукерок немає у продажу, якщо

закрито магазин, якщо забракло грошей тощо - все це не проблеми смаку цукерок, і не можна писати звіт про дефект цукерок виду «цукерки несмачні тому, що закрито магазин».

Деякі автори не дотримуються цієї логіки і допускають у розділі «приготування» помилки. І тут немає «правильного варіанта» — просто в одній традиції вирішено в такий спосіб, в іншій — іншим. У реальній робочій обстановці вам достатньо прочитати кілька тест-кейсів, вже створених вашими колегами, щоб зрозуміти, якої точки зору на це питання вони дотримуються.

Кроки тест-кейсу описують послідовність дій, які необхідно реалізувати у процесі виконання тест-кейсу. Загальні рекомендації щодо написання кроків такі:

починайте зі зрозумілого та очевидного місця, не пишіть зайвих початкових кроків (запуск програми, очевидні операції з інтерфейсом тощо);

навіть якщо в тест-кейсі всього один крок, нумеруйте його (інакше зростає можливість у майбутньому випадково «приклеїти» опис цього кроку до нового тексту);

використовуйте безособову форму (наприклад, "відкрити", "ввести", "додати" замість "відкрийте", "введіть", "додайте"), в англійській мові не треба використовувати частинку "to" (тобто "запустити застосунок" буде «start application», не «to start application»);

співвідносите ступінь деталізації кроків та їх параметрів з метою тест-кейсу, його складністю, рівнем тощо – залежно від цих та багатьох інших факторів ступінь деталізації може змінюватись від загальних ідей до гранично чітко прописаних значень та вказівок;

посилайтеся на попередні кроки та їх діапазони для зменшення обсягу тексту (наприклад, «повторити кроки 3–5 зі значенням…»);

пишіть кроки послідовно, без умовних конструкцій виду «якщо … то…».

Категорично заборонено посилатися на кроки з інших тест-кейсів та на інші тест-кейс повністю: якщо ті, інші тест-кейси будуть змінені або видалені, ваш тест-кейс почне посилатися на невірні дані або в порожнечу, а якщо в процесі виконання ті, інші тест-кейси або кроки приведуть до помилки, ви не зможете закінчити виконання вашого тест-кейсу.

Очікувані результати кожного кроку тест-кейсу описують реакцію програми на дії, описані в полі «кроки тест-кейсу». Номер кроку відповідає номеру результату.

За написанням очікуваних результатів можна порекомендувати:

- описуйте поведінку системи так щоб виключити суб'єктивне тлумачення (наприклад, «застосунок працює вірно» — погано, «з'являється вікно з написом…» — добре);
- пишіть очікуваний результат по всіх кроках без винятку, якщо у вас є хоч найменші сумніви в тому, що результат якогось кроку буде абсолютно тривіальним і очевидним залиште в списку очікуваних результатів порожній рядок - це полегшує сприйняття;
- пишіть коротко, але не на шкоду інформативності;
- уникайте умовних конструкцій виду «якщо … то…».

В очікуваних результатах завжди описується коректна робота програми. Немає і не може бути очікуваного результату у вигляді «застосунок викликає помилку в операційній системі і аварійно завершується зі втратою всіх даних користувача».

При цьому коректна робота програми цілком може припускати відображення повідомлень про невірні дії користувача або деякі критичні ситуації. Так, повідомлення «Неможливо зберегти файл за вказаним шляхом: на цільовому носії недостатньо вільного місця» — це не помилка програми, це абсолютно нормальна і правильна робота. Помилка програми (у цій же ситуації) була б відсутність такого повідомлення, або пошкодження, або втрата даних, що записуються.

3.4. Інструментальні засоби керування тестуванням

Інструментальних засобів управління тестуванням дуже багато, до того ж багато компаній розробляють свої внутрішні засоби вирішення цього завдання.

Немає сенсу заучувати, як працювати з тест-кейсами в тому чи іншому інструментальному засобі — принцип скрізь єдиний, і відповідні навички напрацьовуються буквально за кілька днів. Що на поточний момент важливо розуміти, так це загальний набір функцій, що реалізуються такими інструментальними засобами (звісно, той чи інший

засіб може не реалізовувати якусь функцію з цього списку або реалізувати функції, що не увійшли до списку):

Створення тест-кейсів та наборів тест-кейсів;

Контроль версій документів з можливістю визначити, хто вніс ті чи інші зміни, і скасувати ці зміни, якщо потрібно;

Формування та відстеження реалізації плану тестування, збір та візуалізація різноманітних метрик, генерування звітів;

Інтеграція з системами управління дефектами, фіксація взаємозв'язку між виконанням тест-кейсів та створеними звітами про дефекти;

Інтеграція з системами управління проектами;

Інтеграція з інструментами автоматизованого тестування, управління виконанням автоматизованих тест-кейсів.

Іншими словами, хороший інструментальний засіб управління тестуванням бере на себе всі рутинні технічні операції, які необхідно виконувати в процесі реалізації життєвого циклу тестування. Величезною перевагою також є здатність таких інструментальних засобів відстежувати взаємозв'язки між різними документами та іншими артефактами, взаємозв'язки між артефактами та процесами тощо, підпорядковуючи ці дії системі розмежування прав доступу та гарантуючи збереження та коректність інформації.

Для загального розвитку та кращого закріплення теми про оформлення тест-кейсів розглянемо кілька картинок із формами із різних інструментальних засобів.

Тут цілком свідомо не наведено жодного порівняння чи докладного опису — подібних оглядів достатньо в Інтернеті, і вони стрімко старіють у міру виходу нових версій продуктів, що оглядаються.

Але інтерес представляють окремі особливості інтерфейсу, на які ми звернемо увагу в кожному з прикладів (важливо: якщо вас цікавить докладний опис кожного поля, пов'язаних з ним процесів і т.д. , зверніться до офіційної документації - тут будуть лише короткі пояснення).

3.4.1. Створення тест-кейсу у QAComplete (Рис. 3.3)

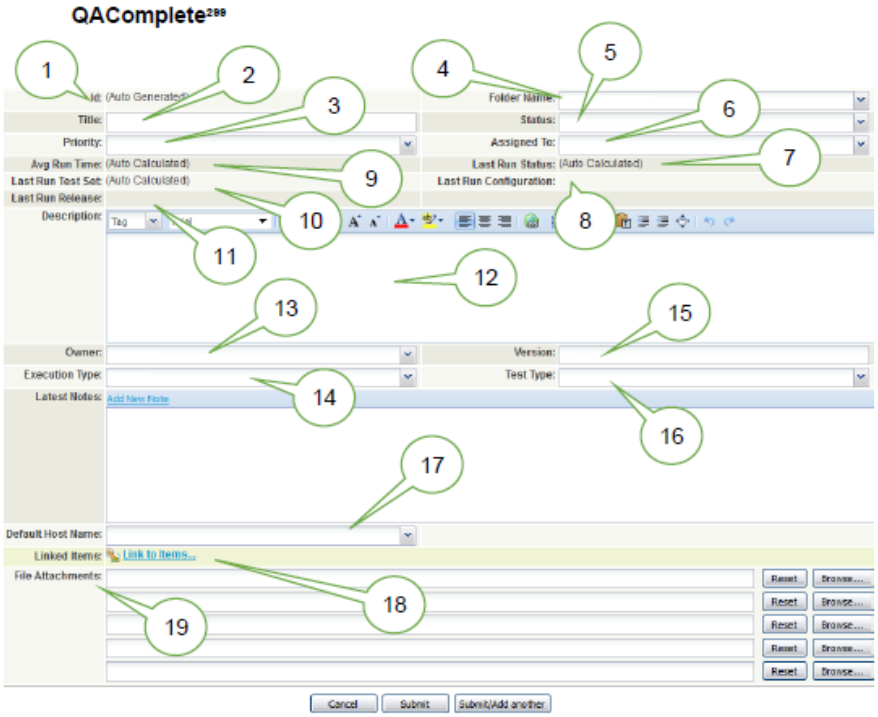

Рисунок 3.3 - Створення тест-кейсу в QAComplete

1. *Id* (ідентифікатор), як видно з відповідного напису, що автогенерується .

2. *Title* (назва), як й у більшості систем, є обов'язковим заповнення.

3. *Priority* (Пріоритет) за замовчуванням пропонує на вибір значення high (високий), medium (середній), low (низький).

4. *Folder name* (розташування) є аналогом полів «Модуль» і «Підмодуль» і дозволяє вибрати з деревоподібного списку, що випадає, відповідне значення, що описує, до чого відноситься тест-кейс.

5. *Status* (статус) показує поточний стан тест-кейсу: new (новий), approved (затверджений), awaiting approval (на розгляді), in design (в розробці), outdated (застарілий), rejected (відхилений).

6. *Assigned to* (виконавець) вказує, хто в даний момент є «основною робочою силою» по даному тест-кейс (або хто має прийняти рішення про, наприклад, затвердження тест-кейсу).

7. *Last Run Status* (результат останнього запуску) показує, чи пройшов тест успішно (passed) чи завершився невдачею (failed).

8. *Last Run Configuration* (Конфігурація, використана для останнього запуску) показує, на якій апаратно-програмній платформі тест-кейс виконувався востаннє.

9. *Avg Run Time* (середній час виконання) містить обчислений автоматично середній час, необхідний виконання тест-кейса.

10. *Last Run Test Set* (востаннє виконувався в наборі) містить інформацію про набір тест-кейсів, в рамках якого тест-кейс виконувався востаннє.

11. *Last Run Release* (востаннє виконувався на випуску) містить інформацію про випуск (білд) програмного засобу, на якому тест-кейс виконувався останній раз.

12. *Description* (опис) дозволяє додати будь-яку корисну інформацію про тест-кейс (включаючи особливості виконання, приготування тощо).

13. *Owner* (власник) вказує на власника тест-кейсу (як правило його автора).

14. *Execution Type* (спосіб виконання) за замовчуванням пропонує лише значення manual (ручне виконання), але за відповідних налаштувань та інтеграції з іншими продуктами список можна розширити (як мінімум додати automated (автоматизоване виконання)).

15. *Version* (версія) містить номер поточної версії тест-кейсу (фактично лічильник того, скільки разів тест-кейс редагували). Вся історія змін зберігається, надаючи можливість повернутися до будь-якої попередньої версії.

16. *Test Type* (вид тесту) за промовчанням пропонує такі варіанти, як negative (негативний), positive (позитивний), regression (регресійний), smoke-test (димовий).

17. *Default host name* (ім'я стандартного хоста) в основному використовується в автоматизованих тест-кейсах і пропонує вибрати зі списку ім'я зареєстрованого комп'ютера, на якому встановлений спеціальний клієнт.

18. *Linked Items* (пов'язані об'єкти) є посилання вимоги, звіти про дефекти тощо .

19. *File Attachments* (вкладення) можуть містити тестові дані, що пояснюють зображення, відеоролики тощо .

Для опису кроків виконання та очікуваних результатів після збереження загального опису тест-кейсу стає доступним додатковий інтерфейс. При необхідності можна додати та налаштувати свої додаткові поля, які значно розширюють вихідні можливості системи.

3.4.2. Створення тест-кейсу в TestLink (Рис. 3.4).

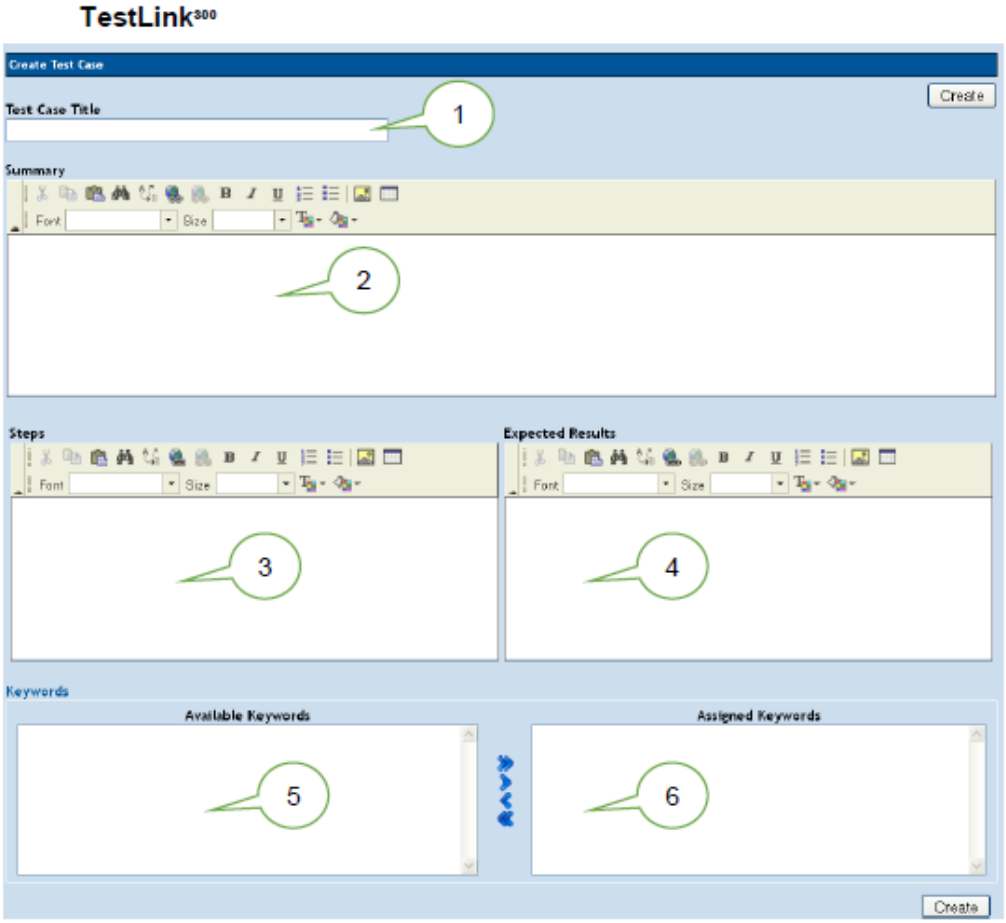

Рисунок 3.4 - Створення тест-кейсу в TestLink

1. *Title* (назва) тут також є обов'язковим для заповнення.

2. *Summary* (опис) дозволяє додати будь-яку корисну інформацію про тест-кейс (включаючи особливості виконання, приготування тощо).

3. *Steps* (Кроки виконання) дозволяє описати кроки виконання.

4. *Expected Results* (очікувані результати) дозволяє описати очікувані результати, що стосуються кроків виконання.

5. *Available Keywords* (доступні ключові слова) містить список ключових слів, які можна порівнювати з тест-кейсом для спрощення класифікації та пошуку тест-кейсів. Це ще одна варіація ідеї «Модулів» та «Підмодулів» (у деяких системах реалізовані обидва механізми).

6. *Assigned Keywords* (призначені ключові слова) містить список ключових слів, асоційованих із тест-кейсом.

Як бачите, інструментальні засоби управління тест-кейсами можуть бути досить мінімалістичними.

3.5. Властивості якісних тест-кейсів

Навіть правильно оформлений тест-кейс може виявитися неякісним, якщо в ньому порушено одну з таких властивостей.

Правильна технічна мова, точність та однаковість формулювань.

Ця властивість однаково стосується і вимог, і тест-кейсів, і звітів про дефекти — будь-якої документації. Основні ідеї можна сформулювати так:

пишіть лаконічно, але зрозуміло;

використовуйте безособову форму дієслів (наприклад, «відкрити» замість «відкрийте»);

обов'язково вказуйте точні імена та технічно вірні назви елементів програми;

не пояснюйте базові принципи роботи з комп'ютером (передбачається, що ваші колеги знають, що таке, наприклад, пункт меню і як з ним працювати);

скрізь називайте ті самі речі однаково (наприклад, не можна в одному тест-кейсі якийсь режим роботи програми назвати «графічне уявлення», а в іншому той же режим — «візуальне відображення», тому що багато людей можуть подумати, що мова йде про різні речі);

дотримуйтесь прийнятого на проекті стандарту оформлення та написання тест-кейсів (іноді такі стандарти можуть бути дуже жорсткими: аж до регламентації того, назви яких елементів мають бути наведені у подвійних лапках, а яких – в одинарних).

Баланс між специфічністю та спільністю. Тест-кейс вважається тим паче специфічним, що більш детально у ньому розписані конкретні дії, конкретні значення тощо, тобто чим більше у ньому конкретики. Відповідно, тест-кейс вважається тим більш загальним, чим у ньому менше конкретики.

Розглянемо поля «кроки» та «очікувані результати» двох тест-кейсів:

Тест-кейс 1:

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: · Створити папки C:/A, C:/B, C:/C, C:/D. · Розмістити в папці C:/D файли 1.html, 2.txt, 3.md з архіву, що додається. 1. Запустити програму, виконавши команду « php converter.php c:/ac:/bc:/c/con-verter.log». 2. Скопіювати файли 1.html, 2.txt, 3.md з папки C:/D до папки C:/A. 3. Зупинити програму натисканням Ctrl+C .	1. Відображається консольний журнал програми з повідомленням поточний час. started , source dir c:/a, destination dir c:/b, log file c:/c/converter.log», у папці C:/C з'являється файл converter.log, у якому з'являється запис « поточний час started , source dir c:/a, destination dir c:/b, log файл c:/c/converter.log». 2. Файли 1.html, 2.txt, 3.md з'являються у папці C:/A, потім зникають звідти і з'являються у папці C:/B. У консольному журналі та файлі C:/C/converter.log з'являються повідомлення (записи) « поточний час processing 1.html (KOI8-R)», « поточний_час processing 2.txt (CP-1251)», « поточний_час processing 3.md (CP-866)». 3. У файлі C:/C/converter.log з'являється запис « поточний час closed ». Програма завершує роботу.

Тест-кейс 2:

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань 1. Виконати конвертацію трьох файлів допустимого розміру в три різні кодування всіх трьох допустимих форматів.	1. Файли переміщуються до папки-приймача, кодування всіх файлів змінюється на UTF-8.

Якщо повернутися до питання «який тест-кейс ви вважали б хорошим, а який — поганим і чому», то відповідь така: обидва тест-кейси погані тому, що перший є надто специфічним, а другий — надто загальним. Можна сказати, що тут до абсурду доведено ідеї низькорівневих та високорівневих тест-кейсів.

Чому погана надмірна специфічність (тест-кейс 1):

при повторних виконаннях тест-кейсу завжди будуть виконуватися строго одні й ті самі дії зі строго одними й тими самими даними, що знижує можливість виявлення помилки;

зростає час написання, доопрацювання і навіть просто прочитання тест-кейсу;

у разі виконання очевидних дій досвідчені фахівці витрачають додаткові розумові ресурси у спробах зрозуміти, що вони пропустили з уваги, так як вони звикли, що так описуються лише найскладніші та неочевидні ситуації.

Чому погана зайва спільність (тест-кейс 2):

тест-кейс складний для виконання тестувальниками-початківцями або навіть досвідченими фахівцями, які лише недавно підключилися до проекту;

недобросовісні співробітники схильні недбало ставитися до таких тест-кейсів;

тестувальник, який виконує тест-кейс, може зрозуміти його інакше, ніж було задумано автором (і в результаті буде виконано практично зовсім інший тест-кейс).

Вихід із цієї ситуації полягає в тому, щоб дотримуватися золотої середини (хоча, звичайно ж, якісь тести будуть трохи специфічнішими, якісь трохи загальнішими). Ось приклад такого серединного підходу:

Тест-кейс 3:

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: · Створити в корені будь-якого диска чотири окремі папки для вхідних файлів, вихідних файлів, файлу журналу та тимчасового зберігання тестових файлів. · Розпакувати вміст архіву в папку для тимчасового зберігання тестових файлів. 1. Запустити програму, вказавши у параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіюйте файли з папки для тимчасового зберігання до папки для вхідних файлів. 3. Зупинити програму.	1. Застосунок запускається і виводить повідомлення про свій запуск в консоль і файл журналу . 2. Файли з папки для вхідних файлів переміщуються у папку для вихідних файлів, у консолі та файлі журналу відображаються повідомлення про конвертацію кожного з файлів із зазначенням його вихідного кодування. 3. Застосунок виводить повідомлення про завершення роботи у файл журналу та завершує роботу.

У цьому тест-кейсі є все необхідне для розуміння та виконання, але при цьому він став коротшим і простішим для виконання, а відсутність строго зазначених значень призводить до того, що при багаторазовому виконанні тест-кейсу (особливо – різними тестувальниками) конкретні параметри змінюватимуть свої значення, що збільшує можливість виявлення помилки.

Ще раз головна думка: самі по собі специфічність чи спільність тест-кейсу не є чимось поганим, але різкий перекіс у той чи інший бік знижує якість тест-кейсу.

Баланс між простотою та складністю. Тут немає академічних визначень, але прийнято вважати, що простий тест-кейс оперує одним об'єктом (чи у ньому явно видно головний об'єкт), і навіть містить небагато тривіальних дій; складний тест-кейс оперує кількома рівноправними об'єктами та містить багато нетривіальних дій.

Переваги простих тест-кейсів:

їх можна швидко прочитати, легко зрозуміти та виконати;

вони зрозумілі початківцям-тестувальникам і новим людям на проекті;

вони роблять наявність помилки очевидною (як правило, у них передбачається виконання повсякденних тривіальних дій, проблеми з якими видно неозброєним поглядом і не викликають дискусій);

вони спрощують початкову діагностику помилки, так як звужують коло пошуку.

Переваги складних тест-кейсів:

при взаємодії багатьох об'єктів підвищується ймовірність виникнення помилки;

користувачі, як правило, використовують складні сценарії, а тому складні тести повноцінніше емулюють роботу користувачів;

програмісти рідко перевіряють такі складні випадки (і вони зовсім не зобов'язані це робити).

Розглянемо приклади.

Занадто простий тест-кейс:

Кроки	Очікувані результати
Запуск програми 1. Запустити програму.	1. Програма запускається.

Занадто складний тест-кейс:

Кроки	Очікувані результати
Повторна конвертація Приготування: Створити в корені будь-якого диска три окремі папки для вхідних файлів, вихідних файлів, файлу журналу. Підготувати набір з декількох файлів максимального розміру підтримуваних форматів з підтримуваними кодуваннями, а також декількох файлів допустимого розміру, але недопустимого формату. 1. Запустити програму, вказавши у параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіюйте в папку для вхідних файлів кілька файлів допустимого формату. 3. Перемістити файли з конвертованих файлів з папки з результуючими файлами до папки для вхідних файлів. 4. Перемістити конвертовані файли з папки з результуючими файлами до папки з набором файлів для тестування. 5. Перемістити всі файли з папки з набором файлів для тестування до папки для вхідних файлів. 6. Перемістити конвертовані файли з папки з результуючими файлами до папки для вхідних файлів. 7. Файли поступово переміщуються з вхідної у вихідну папку, у консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів допустимого формату та повідомлення про ігнорування файлів неприпустимого формату.	2. Файли поступово переміщуються з вхідної у вихідну папку, у консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів. 3. Файли поступово переміщуються з вхідної у вихідну папку, у консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів. 5. Файли поступово переміщуються з вхідної у вихідну папку, у консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів допустимого формату та повідомлення про ігнорування файлів неприпустимого формату.

Цей тест-кейс одночасно є надто складним за надмірністю дій та специфікацією зайвих даних та операцій.

Приклад хорошого складного тест-кейсу може виглядати так:

Кроки	Очікувані результати
Багато копій програми, конфлікт файлових операцій Приготування: Створити в корені будь-якого диска три окремі папки для вхідних файлів, вихідних файлів, файлу журналу. Підготувати набір з декількох файлів максимального розміру підтримуваних форматів з підтримуваними кодуваннями. 1. Запустити першу копію програми, вказавши у параметрах відповідні шляхи із приготування до тесту (ім'я файлу журналу – довільне). 2. Запустіть другу копію програми з тими самими параметрами (див. крок 1). 3. Запустіть третю копію програми з тими самими параметрами (див. крок 1). 4. Змінити пріоритет процесів другої (“high”) та третьої (“low”) копій. 5. Скопіюйте підготовлений набір вихідних файлів до папки для вхідних файлів.	3. Усі три копії програми запускаються, у файлі журналу з'являються послідовно три записи про запуск програми. 5. Файли поступово переміщуються з вхідної у вихідну папку, у консолі та файлі журналу з'являються повідомлення про успішну конвертацію файлів, а також (можливо) повідомлення виду: a. source _ file inaccessible , retrying ”. b. “ destination file inaccessible , retrying ”. c. “ log file inaccessible , retrying ”. Ключовим показником коректної роботи є успішна конвертація всіх файлів, а також поява в консолі та файлі журналу повідомлень про успішну конвертацію кожного файлу (від одного до трьох записів на кожен файл). Повідомлення (попередження) про недоступність вхідного файлу, вихідного файлу або файлу журналу також є показником коректної роботи програми, проте їх кількість залежить від багатьох зовнішніх факторів і не може бути спрогнозовано заздалегідь.

Іноді складніші тест-кейси є також і специфічнішими, але це лише загальна тенденція, а не закон. Також не можна за складністю тест-кейсу однозначно судити про його пріоритет. У нашому прикладі хорошого складного тест-кейсу він явно буде мати дуже низький пріоритет, тому що ситуація, що перевіряється, є штучною і вкрай малоймовірною, але бувають і складні тести з найвищим пріоритетом.

Як і у випадку специфічності та спільності, самі по собі простота або складність тест-кейсів не є чимось поганим (більше того — рекомендується починати розробку та виконання тест-кейсів з простих, а потім переходити до дедалі більш складних), проте надмірна простота і надмірна складність також знижують якість тест-кейсу.

Показовість (висока ймовірність виявлення помилки). Починаючи з рівня тестування критичного шляху, можна стверджувати, що тест-кейс є тим найкращим, чим він більш показовий (з більшою ймовірністю виявляє помилку). Саме тому ми вважаємо непридатними надто прості тест-кейси – вони не показові.

Приклад непоказового (поганого) тест-кейсу:

Кроки	Очікувані результати
Запуск та зупинення програми 1. Запустіть програму з коректними параметрами. 2. Завершити роботу програми.	1. Програма запускається. 2. Програма завершує роботу.

Приклад показового (хорошого) тест-кейсу:

Кроки	Очікувані результати
Запуск із некоректними параметрами, неіснуючі шляхи 1. Запустити програму з усіма трьома параметрами (SOURCE_DIR, DESTINATION_DIR, LOG_FILE_NAME), значення яких вказують на неіснуючі у файловій системі шляхи (наприклад: z:\src\, z:\dst\, z:\log.txt за умови, що у системі немає логічного диска z).	1. У консолі відображаються наведені нижче повідомлення, програма завершує роботу. Повідомлення: a. Повідомлення про використання. b. SOURCE_DIR [z:\ src \]: directory not exists or inaccessible . c. DESTINATION_DIR [z:\ dst \]: directory not exists or inaccessible . d. LOG_FILE_NAME [z:\log.txt]: wrong file name or inaccessible path .

Зверніть увагу, що показовий тест-кейс, як і раніше, залишився досить простим, але він перевіряє ситуацію, виникнення помилки в якій незрівнянно ймовірніше, ніж у ситуації, що описується поганим непоказовим тест-кейсом.

Можна сказати, що показові тест-кейси часто виконують якісь «цікаві дії», тобто такі дії, які навряд чи будуть виконані просто в процесі

роботи з застосунком (наприклад: «зберегти файл» — це звичайна тривіальна дія, яка явно буде виконана не одну сотню разів навіть самими розробниками, а ось «зберегти файл на носій, захищений від запису», «Зберегти файл на носій з недостатнім обсягом вільного простору», «Зберегти файл в папку, до якої немає доступу» - це вже набагато цікавіші та нетривіальніші дії).

Послідовність у досягненні мети. Суть цієї властивості виявляється у тому, що всі дії в тест-кейсі спрямовані слідування єдиної логіці і досягнення єдиної мети і не містять ніяких відхилень.

Прикладами правильної реалізації цієї властивості можуть бути представлені приклади хороших тест-кейсів. А порушення може виглядати так:

Кроки	Очікувані результати
Конвертація з усіх підтримуваних кодувань Приготування: Створити в корені будь-якого диска чотири окремі папки для вхідних файлів, вихідних файлів, файлу журналу та тимчасового зберігання тестових файлів. Розпакувати вміст архіву в папку для тимчасового зберігання тестових файлів. 1. Запустити програму, вказавши у параметрах відповідні шляхи з приготування до тесту (ім'я файлу журналу – довільне). 2. Скопіюйте файли з папки для тимчасового зберігання до папки для вхідних файлів. 3. Зупинити програму. 4. Видалити файл журналу. 5. Повторно запустити програму з тими самими параметрами. 6. Зупинити програму.	1. Програма запускається та виводить повідомлення про свій запуск у консоль та файл журналу. 2. Файли з папки для вхідних файлів переміщуються до папки для вихідних файлів, у консолі та файлі журналу відображаються повідомлення про конвертацію кожного з файлів із зазначенням його вихідного кодування. 3. Програма виводить повідомлення про завершення роботи у файл журналу та завершує роботу. 5. Програма запускається та виводить повідомлення про свій запуск у консоль та заново створений файл журналу. 6. Програма виводить повідомлення про завершення роботи у файл журналу та завершує роботу.

Кроки 3-5 не відповідають меті тест-кейсу, що полягає в перевірці правильності конвертації вхідних даних, представлених у всіх підтримуваних кодуваннях.

Відсутність зайвих дій. Найчастіше ця властивість має на увазі, що не потрібно в кроках тест-кейсу довго і по пунктах розписувати те, що можна замінити однією фразою:

Погано	Добре
1. Вказати як перший параметр програми шлях до папки з вихідними файлами . 2. Вказати як другий параметр програми шлях до папки з кінцевими файлами. 3. Вказати як третій параметр програми шлях до файлу журналу. 4. Запустити програму.	1. Запустити програму з усіма трьома коректними параметрами (наприклад, c:\src\, c:\dst\, c:\log.txt за умови, що відповідні папки існують та доступні застосунку).

Друга за частотою помилка - початок кожного тест-кейсу із запуску програми та докладного опису щодо приведення його в той чи інший стан. У наших прикладах ми розглядаємо кожен тест-кейс як існуючий в єдиному вигляді в ізольованому середовищі, і тому змушені усвідомлено припускатися цієї помилки (інакше тест-кейс буде неповним), але в реальному житті на запуск програми будуть свої тести, а довгий шлях з багатьох дій можна описати як одну дію, з контексту якого зрозуміло, як це дію виконати.

Наступний приклад тест-кейсу не відноситься до нашого "Конвертера файлів", але дуже добре ілюструє цю думку:

Погано	Добре
1. Запустити програму. 2. Вибрати в меню пункт "Файл". 3. Вибрати підпункт "Відкрити". 4. Перейти папку, в якій знаходиться хоча б один файл формату DOCX з трьома і більше сторінками.	1. Відкрити DOCX-файл із трьома та більше сторінками.

І сюди ж можна віднести помилку з повторенням тих самих приготувань у безлічі тест-кейсів. Куди зручніше поєднати тести в набір

і вказати приготування один раз, підкресливши, чи потрібно їх виконувати перед кожним тест-кейсом в наборі.

Проблема з підготовчими (і фінальними) діями ідеально вирішена в автоматизованому модульному тестуванні з використанням фреймворків на кшталт JUnit або TestNG — там існує спеціальний «механізм фіксацій», що автоматично виконує зазначені дії перед кожним окремим тестовим методом (або їх сукупністю).

Ненадмірність по відношенню до інших тест-кейсів. У процесі створення безлічі тест-кейсів дуже легко опинитися в ситуації, коли два і більше тест-кейсів фактично виконують одні і ті ж перевірки, переслідують одні і ті ж цілі, спрямовані на пошук одних і тих же проблем.

Якщо ви виявляєте кілька тест-кейсів, що дублюють завдання один одного, найкраще або видалити все, крім одного, найпоказовішого, або перед видаленням інших на їх основі доопрацювати цей вибраний найпоказовіший тест-кейс.

Демонстративність (здатність демонструвати виявлену помилку очевидним чином). Очікувані результати мають бути підібрані та сформульовані таким чином, щоб будь-яке відхилення від них відразу ж впадало в очі і ставало очевидним, що сталася помилка. Порівняйте витримки із двох тест-кейсів.

Витяг з недемонстративного тест-кейсу:

Кроки	Очікувані результати
5. Розмістити у файлі текст "Приклад довгого тексту, що містить символи українського та англійського алфавіту впереміш." у кодуванні KOI8-R (у слові "Приклад" літера "р" - англійська). 6. Зберегти файл під ім'ям test.txt » і надіслати файл на конвертацію. 7. Перейменувати файл на "test.txt".	6. Програма ігнорує файл. 7. Текст набуває коректного вигляду в кодуванні UTF-8 з урахуванням англійських букв.

У цьому випадку тест-кейс поганий не тільки розпливчастістю формулювання «коректний вигляд у кодуванні UTF-8 з урахуванням

англійських букв», там також дуже легко припуститися помилок при виконанні:

Забути сконвертувати вручну вхідний текст у KOI8-R;

Не помітити, що вперше розширення починається з пропуску;

Забути замінити в слові "Приклад" літеру "р" на англійську;

Через розпливчастість формулювання очікуваного результату прийняти помилкову поведінку, яка виглядає правдоподібно за вірну.

Витяг з демонстративного тест-кейсу:

Кроки	Очікувані результати
5. Розмістити у файлі текст «E╥╔═┼╥ ╘┼═╥╟╘┴.» (Ці символи є словосполучення «Приклад тексту.» у кодуванні KOI8-R, прочитаному як CP866). 6. Надіслати файл на конвертацію.	6. Текст набуває вигляду: «Приклад тексту.» (кодування UTF8).

Другий тест-кейс чітко орієнтований на свою мету по перевірці конвертації (не містить дивної перевірки з ігноруванням файлу з невірним розширенням) і описаний так, що його виконання не становить жодних складнощів, а будь-яке відхилення фактичного результату від очікуваного буде відразу помітно.

Простежуваність. З інформації, що міститься в якісному тест-кейсі, має бути зрозуміло, яку частину програми, які функції і які вимоги він перевіряє. Частково це властивість досягається через заповнення відповідних полів тест-кейсу («Посилання на вимогу», «Модуль», «Підмодуль»), а й сама логіка тест-кейсу грає не останню роль, так як у разі серйозних порушень цієї властивості можна довго з подивом дивитися, наприклад, на яку вимогу посилається тест-кейс, і намагатися зрозуміти, як вони один з одним пов'язані.

Приклад непростежуваного тест-кейсу:

Вимога	Модуль	Підмодуль	Кроки	Очікувані результати
ПТ-4	застосунок		**Поєднання кодувань** Приготування: файл із кількома допустимими та	1. Допустимі кодування конвертуються правильно , неприпустимі

			недопустимими кодуваннями. 1. Надіслати файл на конвертацію.	залишаються без змін.

Так, цей тест-кейс поганий сам по собі (у якісному тест-кейсі складно отримати ситуацію непростежуваності), але в ньому є і особливі недоліки, що ускладнюють простежуваність:

Посилання на неіснуючу вимогу (вимоги ПТ-4 немає).

У полі "Модуль" вказано значення "Застосунок" (за великим рахунком можна було залишати це поле порожнім - це було б так само інформативно), поле "Підмодуль" не заповнено.

За назвою та кроками можна припустити, що цей тест-кейс найближчий до ДС-5.1 і ДС-5.3, але сформульований очікуваний результат не випливає явно з цих вимог.

Приклад тест-кейсу, що простежується:

Вимога	Модуль	Підмодуль	Кроки	Очікувані результати
ДС-2.4, ДС-3.2	Стартер	Обробник помилок	**Запуск із некоректними параметрами, неіснуючі шляхи** 1. Запустити програму з усіма трьома параметрами, значення яких вказують на шляхи, що не існують у файловій системі.	1. У консолі відображаються наведені нижче повідомлення, програма завершує роботу. Повідомлення a. SOURCE_DIR [шлях]: directory not exists or inaccessible . b. DESTINATION_DIR [шлях]: directory not exists or inaccessible . c. LOG_FILE_NAME [ім'я]: wrong file name or inaccessible path .

Можна подумати, що цей тест-кейс зачіпає ДС-2 і ДС-3 цілком, але в полі «Вимога» є цілком чітка конкретизація, до того ж зазначені модуль, підмодуль і сама логіка тест-кейсу усувають сумніви.

Деякі автори також підкреслюють, що простежуваність тест-кейс пов'язана з його надмірністю по відношенню до інших тест-кейс (набагато простіше дати посилання на один унікальний тест-кейс, ніж вибирати з кількох дуже схожих).

Можливість повторного використання. Ця властивість рідко виконується для низькорівневих тест-кейсів, але при створенні високорівневих тест-кейсів можна досягти таких формулювань, при яких:

Тест-кейс буде придатним до використання з різними налаштуваннями тестованого застосунка та в різних тестових оточеннях;

Тест-кейс практично без змін можна буде використовувати для тестування аналогічної функціональності в інших проектах або інших галузях програми.

Прикладом тест-кейсу, який важко використовувати повторно, може бути практично будь-який тест-кейс із високою специфічністю.

Не найідеальнішим, але дуже наочним прикладом тест-кейсу, який може бути легко використаний у різних проектах, може служити наступний тест-кейс:

Кроки	Очікувані результати
Запуск, всі параметри некоректні 1. Запустити програму, вказавши як всі параметри свідомо некоректні значення.	1. Програма запускається, після чого виводить повідомлення з описом суті проблеми з кожним із параметрів та завершує роботу.

Повторюваність. Тест-кейс має бути сформульований таким чином, щоб при багаторазовому повторенні він показував однакові результати. Цю властивість можна розділити на два підпункти:

по-перше, навіть загальні формулювання, що допускають різні варіанти виконання тест-кейсу, повинні окреслювати відповідні явні межі (наприклад: «ввести якесь число» - погано, «ввести ціле число в діапазоні від -273 до +500 включно» - добре);

по-друге, дії (кроки) тест-кейсу по можливості не повинні призводити до незворотних (або складно оборотних) наслідків (наприклад: видалення даних, порушення конфігурації оточення тощо) - не варто включати до тест-кейсу такі «руйнівні дії», якщо вони явно не

продиктовані метою тест-кейсу; якщо ж мета тест-кейсу зобов'язує нас до виконання таких дій, у самому тест-кейсі має бути опис дій щодо відновлення вихідного стану програми (даних, оточення).

Відповідність прийнятим шаблонам оформлення та традиціям. З шаблонами оформлення, як правило, проблем не виникає: вони суворо визначені зразком або взагалі екранною формою інструментального засобу управління тест-кейсами. Що ж до традицій, то вони відрізняються навіть у різних командах у рамках однієї компанії, і тут неможливо дати іншої поради, окрім як «шануйте вже готові тест-кейси перед тим як писати свої».

У разі обійдемося без окремих прикладів, т.к. вище і так наведено багато правильно оформлених тест-кейсів.

3.6. Набори тест-кейсів

3.6.1. Термінологія та загальні відомості

Набір тест-кейсів — сукупність тест-кейсів, вибраних із певною загальною метою або за деякою загальною ознакою. Іноді в такій сукупності результати завершення одного тест-кейсу стають вхідним станом програми наступного тест-кейсу.

Як ми тільки що переконалися на прикладі безлічі окремих тест-кейсів, вкрай незручно (більше того, це помилка!) щоразу писати в кожному тест-кейсі одні й самі приготування і повторювати одні й самі початкові кроки.

Набагато зручніше об'єднати кілька тест-кейсів у набір чи послідовність. І тут ми приходимо до класифікації наборів тест-кейсів.

У загальному випадку набори тест-кейсів можна розділити на вільні (порядок виконання тест-кейсів не важливий) та послідовні (порядок виконання тест-кейсів важливий).

Переваги вільних наборів:

Тест-кейси можна виконувати у будь-якому зручному порядку, а також створювати «набори всередині наборів».

Якщо якийсь тест-кейс завершився помилкою, це не вплине на можливість виконання інших тест-кейсів.

Переваги послідовних наборів:

Кожен наступний у наборі тест-кейс як вхідний стан програми отримує результат роботи попереднього тест-кейсу, що дозволяє сильно скоротити кількість кроків в окремих тест-кейсах.

Довгі послідовності дій куди краще імітують роботу реальних користувачів, ніж окремі «точкові» на застосунок.

3.6.2. Користувальницькі сценарії (сценарії використання).

До окремого підвиду послідовних наборів тест-кейсів (або навіть неоформлених ідей тест-кейсів, таких, як пункти чек-листа) можна віднести користувальницькі сценарії, що є ланцюжками дій, що виконуються користувачем у певній ситуації для досягнення певної мети.

В даному випадку мова не йде про use cases (варіанти використання), що є формою вимог. Користувальницькі сценарії як техніка тестування куди менш формалізовані, хоча можуть будуватися з урахуванням варіантів використання.

Пояснимо це спочатку на прикладі, що не стосується «Конвертера файлів». Допустимо, користувач хоче роздрукувати табличку на двері кабінету з текстом «Йде робота, не стукати!» Для цього йому потрібно:

1) Запустити текстовий редактор.

2) Створити новий документ (якщо редактор не робить це самостійно).

3) Набрати у документі текст.

4) Відформатувати текст належним чином.

5) Надіслати документ на друк.

6) Зберегти документ (спірно, але допустимо).

7) Закрити текстовий редактор.

Ось ми і отримали сценарій користувача, пункти якого можуть стати основою для кроків тест-кейсу або цілого набору окремих тест-кейсів.

Сценарії можуть бути досить довгими і складними, можуть містити в собі цикли і умовні розгалуження, але при всьому цьому вони мають ряд дуже цікавих переваг:

Сценарії показують реальні та зрозумілі приклади використання продукту (на відміну від великих чек-листів, де сенс окремих пунктів може губитися).

Сценарії зрозумілі кінцевим користувачам і добре підходять для обговорення та спільного покращення.

Сценарії та їх частини легше оцінювати з погляду важливості, ніж окремі пункти (особливо низькорівневих) вимог.

Сценарії відмінно показують недоробки у вимогах (якщо стає незрозуміло, що робити в тому чи іншому пункті сценарію, - з вимогами явно щось не те).

У граничному випадку (брак часу та інші форс-мажори) сценарії можна навіть не прописувати докладно, а просто називати - і саме найменування вже підкаже досвідченому фахівцю, що робити.

3.6.3. Детальна класифікація наборів тест-кейсів.

Детальна класифікація наборів тест-кейс може бути виражена наступною таблицею (табл. 3.1).

Таблиця 3.1 - Детальна класифікація наборів тест-кейсів

		За ізольованістю тест-кейсів один від одного	
		Ізольовані	Узагальнені
За освітою тест-кейсами суворої послідовності	Вільні	Ізольовані вільні	Узагальнені вільні
	Послідовні	Ізольовані послідовні	Узагальнені послідовні

Набір ізольованих вільних тест-кейсів: дії розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейс можна виконувати в будь-якому порядку.

Набір узагальнених вільних тест-кейсів: дії з розділу приготування потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси можна виконувати в будь-якому порядку.

Набір ізольованих послідовних тест-кейсів: дії з розділу «приготування» потрібно повторити перед кожним тест-кейсом, а самі тест-кейс потрібно виконувати в строго визначеному порядку.

Набір узагальнених послідовних тест-кейсів: дії з розділу приготування потрібно виконати один раз (а потім просто виконувати тест-кейси), а самі тест-кейси потрібно виконувати в строго визначеному порядку.

Головна перевага ізольованості: кожен тест-кейс виконується у «чистому середовищі», на нього не впливають результати роботи попередніх тест-кейсів.

Головна перевага узагальненості: приготування не потрібно повторювати (економія часу).

Головна перевага послідовності: відчутне скорочення кроків у кожному тест-кейсі, так як результат виконання попереднього тест-кейсу є початковою ситуацією для наступного.

Головна перевага свободи: можливість виконувати тест-кейси в будь-якому порядку, а також те, що при провалі якогось тест-кейсу (застосунок не прийшов в очікуваний стан) інші тест -кейси, як і раніше, можна виконувати.

3.6.4. Принципи побудови наборів тест-кейсів.

Тепер про найголовніше: як формувати набори тест-кейсів. Правильна відповідь звучить дуже коротко: логічно. І це не жарт. Єдине завдання наборів — підвищити ефективність тестування за рахунок прискорення та спрощення виконання тест-кейсів, збільшення глибини дослідження якоїсь галузі програми або функціональності, дотримання типових користувальницьких сценаріїв або зручної послідовності виконання тест-кейсів і т.д.

Набір тест-кейсів завжди створюється з якоюсь метою, на основі якоїсь логіки, і за цими ж принципами в набір включаються тести, що мають відповідні властивості.

Якщо ж говорити про найбільш типові підходи до складання наборів тест-кейсів, можна позначити наступне:

На основі чек-аркушів. Подивіться уважно на приклади чек-листів, які ми розробили у відповідному розділі: кожен пункт чек-листа може перетворитися на кілька тест-кейсів — і ми отримуємо готовий набір.

На основі розбиття програми на модулі та підмодулі. Для кожного модуля (або окремих підмодулів) можна скласти свій набір тест-кейсів.

За принципом перевірки найважливіших, менш важливих та інших функцій докладання (саме у цьому принципу ми становили приклади чек-листів).

За принципом угруповання тест-кейсів для перевірки певного рівня вимог чи типу вимог, групи вимог чи окремої вимоги.

За принципом частоти виявлення тест-кейсами дефектів у застосунку (наприклад, бачимо, деякі тест-кейси щоразу завершуються невдачею, отже, ми можемо об'єднати в набор, умовно названий «проблемні місця у застосунку»).

За архітектурним принципом: набори для перевірки інтерфейсу користувача і всього рівня представлення, для перевірки рівня бізнес-логіки, для перевірки рівня даних.

По області внутрішньої роботи програми, наприклад: «тест-кейси, що стосуються роботи з базою даних», «тест-кейси, що стосуються роботи з файловою системою», «тест-кейси, що стосуються роботи з мережею», і т.д.

Не потрібно заучувати цей перелік. Це лише приклади — грубо кажучи, «перше, що спадає на думку». Важливим є принцип: якщо ви бачите, що виконання деяких тест-кейсів у вигляді набору принесе вам користь, створюйте такий набір. Примітка: без хороших інструментальних засобів управління тест-кейс працювати з наборами тест-кейсів вкрай важко, так як доводиться самостійно стежити за приготуваннями, «недостающими кроками», ізольованістю чи узагальненістю, вільністю чи послідовністю тощо.

3.7. Логіка створення ефективних перевірок

Тепер, коли ми розглянули принципи побудови чек-листів та оформлення тест-кейсів, властивості якісних тест-кейсів, а також принципи об'єднання тест-кейсів у набори, настав час перейти до найскладнішої, «філософської» частини, в якій ми будемо вже міркувати не про те, що і як писати, а про те, як думати.

Раніше вже було сказано: якщо у тест-кейсу не вказані вхідні дані, умови виконання та очікувані результати, або не зрозуміла мета тест-кейсу – це поганий тест-кейс. І тут на чільному місці стоїть мета. Якщо ми чітко розуміємо, що і навіщо ми робимо, ми або швидко знаходимо всю решту інформації, що бракує, або так само швидко формулюємо правильні питання і адресуємо їх правильним людям.

Вся суть роботи тестувальника зрештою спрямовано на підвищення якості (процесів, продуктів тощо). Але що таке якість? Існує сухе офіційне визначення, але навіть там сказано про «потреби та очікування користувача (замовника)».

І тут проявляється головна думка: *якість - це якась цінність для кінцевого користувача (замовника)*. Людина у будь-якому разі платить за використання продукту — грошима, своїм часом, якимись зусиллями. Навіть якщо ви не отримуєте цю «оплату», людина цілком обґрунтовано вважає, що щось уже на вас витратила, і вона має рацію. Але чи отримує вона те, на що розраховувала (припустимо, що її очікування здорові та реалістичні)?

Якщо ми підходимо до тестування формально, ми ризикуємо отримати продукт, який за документами (метриками тощо) виглядає ідеально, але насправді нікому не потрібен.

Оскільки практично будь-який сучасний програмний продукт є непростою системою, серед безлічі її властивостей і функцій об'єктивно є найважливіші, менш важливі і зовсім незначні для користувачів.

Якщо зусилля тестувальників будуть сконцентровані на першій і другій категорії (найважливішому і менш важливому), наші шанси створити застосунок, що задовольняє замовника, різко збільшуються.

Є проста логіка:

1. Тести шукають помилок.
2. Але всі помилки знайти неможливо.
3. Отже, наше завдання знайти максимум ВАЖЛИВИХ помилок за наявний час.

Під важливими помилками ми розуміємо такі, які призводять до порушення важливих для користувача функцій або властивостей продукту. Функції та властивості розділені не випадково – безпека, продуктивність, зручність тощо не відносяться до функцій, але відіграють не менш важливу роль у формуванні задоволеності замовника та кінцевих користувачів.

Ситуація посилюється такими фактами:

через безліч економічних і технічних причин ми не можемо виконати «всі тести, що прийдуть нам на думку» (та ще й багаторазово) — тут доводиться ретельно вибирати, що і як ми тестуватимемо, беручи до уваги щойно згадану думку: якість — це певна цінність для кінцевого користувача (замовника);

ніколи в реальному житті (хоч би як ми намагалися) ми не отримаємо «досконалого та ідеального набору вимог» — там завжди буде деяка кількість недоробок, і це теж треба брати до уваги.

Однак існує досить простий алгоритм, що дозволяє нам створювати ефективні перевірки навіть за таких умов. Приступаючи до продумування чек-листа, тест-кейс або набору тест-кейсів, задайте собі такі запитання та отримайте чіткі відповіді:

Що перед вами? Якщо ви не розумієте, що вам належить тестувати, ви не підете далі бездумних формальних перевірок.

Кому і навіщо воно потрібне (і наскільки це важливо)? Відповідь на це питання дозволить вам швидко придумати кілька характерних сценаріїв використання того, що ви збираєтесь тестувати.

Як воно зазвичай використовується? Це вже деталізація сценаріїв та джерело ідей для позитивного тестування (їх зручно оформити у вигляді чек-листа).

Як може зламатися, тобто почати працювати неправильно? Це також деталізація сценаріїв використання, але вже у контексті негативного тестування (їх також зручно оформити у вигляді чек-листа).

До цього алгоритму можна додати ще невеликий перелік універсальних рекомендацій, які дозволять вам проводити тестування краще:

Починайте якомога раніше - вже з моменту появи перших вимог можна займатися їх тестуванням та покращенням, можна писати чек-листи та тест-кейси, можна уточнювати план тестування, готувати тестове оточення тощо.

Якщо ви маєте тестувати щось велике і складне, розбивайте його на модулі та підмодулі, функціональність. Піддавайте функціональній декомпозиції - тобто досягайте такого рівня деталізації, при якому ви можете легко пам'ятати всю інформацію про об'єкт тестування.

Обов'язково пишіть чек-листи. Якщо вам здається, що ви зможете запам'ятати всі ідеї і потім легко відтворити їх, ви помиляєтеся. Винятків не буває.

У міру створення чек-листів, тест-кейсів і т.д. прямо в текст вписуйте питання, що виникають. Коли питань накопичиться достатньо, зберіть їх окремо, уточніть формулювання та зверніться до того, хто може дати відповіді.

Якщо інструментальний засіб, що використовується вами, дозволяє використовувати косметичне оформлення тексту - використовуйте (так текст буде краще читатися), але намагайтеся

слідувати загальноприйнятим традиціям і не розфарбовувати кожне друге слово у свій колір, шрифт, розмір і т.д.

Використовуйте техніку швидкого перегляду для отримання відгуку від колег та покращення створеного вами документа.

Плануйте час на покращення тест-кейсів (виправлення помилок, доопрацювання за фактом зміни вимог тощо).

Починайте опрацювання (і виконання) тест-кейсів із простих позитивних перевірок найважливішої функціональності. Потім поступово підвищуйте складність перевірок, пам'ятаючи не лише про позитивні, а й негативні перевірки.

Пам'ятайте, що в основі тестування лежить ціль. Якщо ви не можете швидко і просто сформулювати ціль створеного вами тест-кейсу, ви створили поганий тест-кейс.

Уникайте надлишкових, дублюючих один одного тест-кейсів. Мінімізувати їхню кількість вам допоможуть техніки класів еквівалентності, граничних умов, доменного тестування.

Якщо показник тест-кейсу можна збільшити, при цьому не сильно змінивши його складність і не відхилившись від вихідної мети, зробіть це.

Пам'ятайте, що багато тест-кейси вимагають окремої підготовки, яку потрібно описати у відповідному полі тест-кейсу.

Декілька позитивних тест-кейсів можна безбоязно об'єднувати, але об'єднання негативних тест-кейсів майже завжди заборонено.

Подумайте, як можна оптимізувати створений вами тест-кейс (набір тест-кейсів тощо) так, щоб знизити трудовитрати на його виконання.

Перед тим як відправляти фінальну версію створеного вами документа, ще раз перечитайте написане (у добрій половині випадків знайдете помилку або іншу недоробку).

3.8. Приклад реалізації логіки створення ефективних перевірок

Раніше ми склали докладний чек-лист для тестування нашого конвертера файлів. Давайте подивимося на нього критично і подумаємо: що можна скоротити, чим ми пожертвуємо і який виграш отримаємо.

Перш ніж ми почнемо оптимізувати чек-лист, важливо зазначити, що рішення про те, що важливо і неважливо, варто приймати на основі ранжування вимог щодо важливості, а також узгоджувати із замовником.

Що для користувача найважливіше? Заради чого створювався застосунок? Щоб конвертувати файли. Зважаючи на те, що налаштування програми виконуватиме кваліфікований технічний фахівець, ми можемо навіть «відсунути на другий план» реакцію програми на помилки стадії запуску та завершення.

І на перше місце виходить:

Обробка файлів, різні формати, кодування та розміри:

Таблиця 3.2 - Формати, кодування та розміри файлів

		Формати вхідних файлів		
		TXT	HTML	MD
Кодування вхідних файлів	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ
	Будь-яка	0 байт		
	Будь-яка	50 МБ + 1 Б	50 МБ + 1 Б	50 МБ + 1 Б
	-	Будь-який неприпустимий формат		
	Будь-яка	Пошкодження у допустимому форматі		

Чи можемо ми прискорити виконання цих перевірок (адже їх багато)? Можемо. І у нас навіть є два взаємодоповнюючі інструменти:

подальша класифікація за важливістю;

автоматизація тестування.

Спочатку поділимо таблицю на дві — трохи важливіше і менш важливе. У «трохи важливіше» потрапляє:

Таблиця 3.3 - Формати, кодування та розміри файлів

		Формати вхідних файлів		
		TXT	HTML	MD
Кодування вхідних файлів	WIN1251	100 КБ	50 МБ	10 МБ
	CP866	10 МБ	100 КБ	50 МБ
	KOI8R	50 МБ	10 МБ	100 КБ

Підготуємо 18 файлів - 9 вихідних + 9 сконвертованих (у будь-якому текстовому редакторі з функцією конвертації та кодувань), щоб надалі порівнювати роботу нашої програми з еталоном.

Для «трохи менш важливого» залишилося:

Файл розміром 0 байт (об'єктивно йому не важлива така характеристика, як «кодування»). Підготуємо один файл розміром 0 байт.

Файл розміром 50 МБ +1Б (для нього теж не важливе кодування). Підготуємо один файл розміром 52 428 801 байт.

Будь-який неприпустимий формат:

- По розширенню (файл з розширенням, відмінним від . txt , . html , . md). Беремо будь-який довільний файл, наприклад картинку (розмір від 1 до 50 МБ, розширення .jpg) .
- За внутрішнім змістом (наприклад, . jpg перейменували на . txt). Копії файлу з попереднього пункту даємо розширення. txt .

Пошкодження у допустимому форматі. Викреслюємо. Взагалі. Навіть вкрай складні дорогі редактори далеко не завжди здатні відновити пошкоджені файли своїх форматів, а наша програма — це мініатюрна утиліта конвертації кодувань, і не варто чекати від неї можливостей професійного інструментального засобу відновлення даних.

Що ми отримали у результаті? Нам потрібно підготувати наступні 22 файли (оскільки файли все одно мають імена, посилимо цей набір тестових даних, представивши в іменах файлів символи латиниці, кирилиці та спецсимволи).

Таблиця 3.3 – Підсумковий набір файлів для тестування програми

№	Ім'я	Кодування	Розмір
1	"Дрібний" файл у WIN1251.txt	WIN1251	100 КБ
2	"Середній" файл в CP866.txt	CP866	10 МБ
3	«Великий» файл у KOI8R.txt	KOI8R	50 МБ
4	"Великий" файл у win-1251.html	WIN1251	50 МБ
5	"Дрібний" файл в cp-866.html	CP866	100 КБ
6	"Середній" файл в koi8-r.html	KOI8R	10 МБ
7	"Середній" файл у WIN_1251.md	WIN1251	10 МБ
8	"Великий" файл в CP_866.md	CP866	50 МБ
9	"Дрібний" файл в KOI8_R.md	KOI8R	100 КБ
10	"Дрібний" еталон WIN1251.txt	UTF8	100 КБ
11	"Середній" еталон CP866.txt	UTF8	10 МБ
12	«Великий» стандарт KOI8R.txt	UTF8	50 МБ
13	«Великий» еталон у win-1251.html	UTF8	50 МБ
14	"Дрібний" еталон в cp-866.html	UTF8	100 КБ

15	"Середній" еталон в koi8-r.html	UTF8	10 МБ
16	"Середній" еталон у WIN_1251.md	UTF8	10 МБ
17	«Великий» стандарт в CP_866.md	UTF8	50 МБ
18	«Дрібний» еталон у KOI8_R.md	UTF8	100 КБ
19	Порожній файл.md	-	0 Б
20	Занадто великий файл.txt	-	52'428'801 Б
21	Картинка.jpg	-	~ 1 МБ
22	Зображення у вигляді TXT.txt	-	~ 1 МБ

Ми згадували автоматизацію як спосіб прискорення виконання тест-кейсів. В даному випадку ми можемо обійтися найтривіальнішими командними файлами. Спеціально створені скрипти дозволяють повністю автоматизувати виконання всього рівня димового тестування над представленим набором з 22 файлів.

Якщо знову повернутись до чек-листа, то виявляється, що ми вже підготували перевірки для всього рівня димового тестування та частини рівня тестування критичного шляху.

Продовжимо оптимізацію. Більшість перевірок не становлять особливої складності, і ми розберемося з ними по ходу справи, але є в чек-листі пункт, що викликає особливу тривогу: продуктивність.

Тестування та оптимізація продуктивності - це окремий вид тестування зі своїми досить непростими правилами та підходами, до того ж він поділяється на кілька підвидів. Чи потрібна це нам у нашому застосунку?

Замовник в АК-1.1 визначив мінімальну продуктивність програми як здатність обробляти вхідні дані зі швидкістю щонайменше 5 МБ/сек. Грубі експерименти на вказаному в АК-1.1 обладнанні показують, що навіть набагато складніші операції (наприклад, архівування відеофайлу з максимальним ступенем стиснення) виконуються швидше (нехай і ненабагато). Висновок? Викреслюємо. Імовірність зустріти тут проблему дуже мала, а відповідне тестування вимагає відчутних витрат сил і часу, а також наявності відповідних фахівців.

Перепишемо компактно те, що залишилося від рівня тестування критичного шляху. Це ще не тест-кейс! Це лише одна форма запису чек-листа, зручніша цьому етапі.

Таблиця 3.4 – Чек-лист для рівня критичного шляху

Суть перевірки	Очікувана реакція
Запуск параметрів.	Відображення інструкції щодо використання.
Запуск із недостатньою кількістю параметрів.	Відображення інструкції щодо використання та вказівки імен відсутніх параметрів.
Запуск із неправильними значеннями параметрів: • Неприпустимий шлях SOURCE_DIR. • Неприпустимий шлях DESTINATION_DIR. • Неприпустиме ім'я LOG_FILE_NAME. • DESTINATION_DIR знаходиться всередині SOURCE_DIR. • Значення DESTINATION_DIR та SOURCE_DIR збігаються.	Відображення інструкції щодо використання та вказівки імені неправильного параметра, значення неправильного параметра та пояснення суті проблеми.
Недоступні вхідні файли: • Немає прав доступу. • Файл відкритий та заблокований. • Файл із атрибутом "тільки для читання".	Відображення повідомлення в консоль та файл журналу, подальше ігнорування недоступних файлів.
Журнал роботи програми: • Автоматичне створення (за відсутності журналу) ім'я журналу вказано явно. • Продовження (доповнення журналу) під час повторних запусків, ім'я журналу не вказано.	Створення або продовження ведення файлу журналу за вказаним чи обчисленим шляхом.

Зрештою, у нас залишився рівень розширеного тестування. І зараз ми зробимо те, чого за всіма класичними книгами вчать не робити — ми відмовимося від усього цього набору перевірок цілком.

Ми зараз справді підвищили ризик пропустити якийсь дефект. Але ймовірність виникнення такого дефекту мала через малу ймовірність виникнення описаних у цих перевірках ситуацій. При цьому за найскромнішими підрахунками ми на третину скоротили загальну кількість перевірок, які нам потрібно буде виконувати, а значить —

вивільнили сили та час для більш ретельного опрацювання типових щоденних сценаріїв використання програми.

Весь оптимізований чек-лист (він же — чернетка для плану виконання перевірок) тепер виглядає так:

1) Підготувати файли (табл. 3.3 - Підсумковий набір файлів для тестування програми).

2) Для "димового тесту" використовувати командні файли.

3) Для основних перевірок використовувати файли з пункту 1 та такі ідеї (табл. 3.4 - Чек-лист для рівня критичного шляху).

4) У разі наявності часу використовувати початкову редакцію чек-листа для рівня розширеного тестування як основу для виконання дослідницького тестування.

І майже все. Залишається тільки ще раз підкреслити, що представлена логіка вибору перевірок не претендує на те, щоб вважатися єдино вірною, але вона явно дозволяє заощадити величезну кількість зусиль, при цьому практично не знизивши якість тестування функціональності програми, що найбільш затребувана замовником.

3.9. Типові помилки при розробці чек-листів, тест-кейсів та наборів тест-кейсів

Помилки оформлення та формулювань.

Відсутність назви тест-кейсу або погано написана назва. В більшості систем управління тест-кейсами поле для назви винесено окремо і є обов'язковим до заповнення - тоді ця проблема відпадає. Якщо інструментальний засіб дозволяє створити тест-кейс без назви, виникає ризик отримати N тест-кейсів, для розуміння суті кожного з яких потрібно прочитати десятки рядків замість однієї пропозиції. Це гарантоване вбивство робочого часу та зниження продуктивності команди на порядок.

Якщо назва тест-кейсу доводиться вписувати в полі з кроками і інструментальний засіб допускає форматування тексту, назву варто писати жирним шрифтом, щоб його було легше відокремлювати від основного тексту.

Продовженням цієї помилки є створення однакових назв, якими об'єктивно неможливо відрізнити один тест-кейс від іншого. Більше того,

виникає підозра, що однаково озаглавлені тест-кейси і всередині однакові. Тому слід формулювати назви по-різному, підкреслюючи суть тест-кейса та її відміну від інших, схожих тест-кейсів.

І, нарешті, у назві неприпустимі «сміттєві слова» типу «перевірка», «тест» тощо. Адже це назва тест-кейсу, тобто мова щодо визначення йде про перевірку, і не треба цей факт підкреслювати додатково.

Відсутність нумерації кроків або очікуваних результатів (навіть якщо така лише один). Наявність цієї помилки перетворює тест-кейс на «потік свідомості», в якому немає структурованості, модифікованості та інших корисних властивостей — стає дуже легко переплутати, що до чого стосується. Навіть виконання такого тест-кейсу ускладнюється, а доопрацювання взагалі перетворюється на каторжну працю.

Посилання на безліч вимог. Іноді високорівневий тест-кейс дійсно зачіпає кілька вимог, але в такому разі рекомендується писати посилання на максимум 2–3 найголовніших (найбільш відповідних цілей тест-кейсу), а ще краще – вказувати загальний розділ цих вимог (тобто не посилатися, наприклад, на вимоги 5.7.1, 5.7.2, 5.7.3, 5.7.7, 5.7.9, 5.7.12, а просто послатися на розділ 5.7, що включає всі перелічені пункти). У більшості інструментальних засобів управління тест-кейсами це поле є список, що випадає, і там ця проблема втрачає актуальність.

Використання особистої форми дієслів. Рекомендується писати "натиснути" замість "натисніть", "ввести" замість "введіть", "перейти" замість "перейдіть" і т.д. У технічній документації взагалі не рекомендується «переходити на особистості», а також існує думка, що особиста форма дієслів підсвідомо сприймається як «чужі безглузді команди» та призводить до підвищеної стомлюваності та дратівливості.

Використання минулого чи майбутнього часу в очікуваних результатах. Це не дуже серйозна помилка, але все одно "введене значення відображається в полі" читається краще, ніж "введене значення буде відображатися в полі".

Постійне використання слів "перевірити" (і йому подібних) у чек-листах. У результаті майже кожен пункт чек-листа починається з

«перевірити…», «перевірити…», «перевірити…». Але ж весь чек-лист — це список перевірок! Навіщо писати це слово? Сюди відноситься типове слово «спробувати» в негативних тест-кейсах («спробувати поділити на нуль», «спробувати відкрити неіснуючий файл», «спробувати ввести неприпустимі символи»). Треба писати: «поділ на нуль», «відкриття неіснуючого файлу», «введення спецсимволів». Так набагато коротше та інформативніше. А реакцію програми, якщо вона не очевидна, можна вказати в дужках (так буде навіть інформативніше): «поділ на нуль» (повідомлення «Division by zero detected»), «відкриття неіснуючого файлу» (приводить до автоматичного створення файлу), «введення спецсимволів» (символи не вводяться, відображається підказка).

Опис стандартних елементів інтерфейсу замість використання їх усталених назв. «Маленький хрестик праворуч у верхній частині вікна програми» — це системна кнопка «Закрити» (system button "Close"), "швидко-швидко двічі натиснути на ліву клавішу миші" - це подвійне клацання (double click), «маленьке віконце з написом з'являється, коли наводиш мишу» - це підказка (hint).

Пунктуаційні, орфографічні, синтаксичні та подібні до них помилки. Без коментарів.

Логічні помилки

Посилання на інші тест-кейс або кроки інших тест-кейсів. За винятком випадків написання строго обумовленого явно позначеного набору послідовних тест-кейсів, це заборонено робити. У найкращому разі вам пощастить, і тест-кейс, на який ви посилалися, буде просто віддалено — пощастить тому, що це буде одразу помітно. Не пощастить у випадку, якщо тест-кейс, на який ви посилаєтеся, буде змінено - посилання, як і раніше, веде в якесь існуюче місце, але описано там вже зовсім не те, що було в момент створення посилання.

Деталізація, що не відповідає рівню функціонального тестування. Наприклад, не потрібно на рівні димового тестування перевіряти працездатність кожної окремої кнопки або прописувати якийсь вкрай складний, нетривіальний і рідкісний сценарій — поведінка

кнопок і без явної вказівки буде перевірена безліччю тест-кейсів, які об'єктивно задіють ці кнопки, а складному сценарію місце на рівні тестування критичного шляху або навіть на рівні розширеного тестування (у яких, навпаки, недоліком вважатимуться зайве узагальнення без належної деталізації).

Розпливчасті двозначні описи дій та очікуваних результатів. Пам'ятайте, що тест-кейс з високою ймовірністю виконуватимете не ви (автор тест-кейсу), а інший співробітник, і він не телепат. Спробуйте здогадатися за цими прикладами, що мав на увазі автор:

"Встановити програму на диск С". (Тобто в «C:\»? Прямо в корінь? Або як?)

"Натиснути на піктограму програми". (Наприклад, якщо у мене є ico-файл з іконкою програми, і я по ньому клікну це воно? Чи ні?)

"Вікно програми запуститься". (Куди?)

«Працює правильно». (А вірно - це, вибачте, як?)

"OK". (І? Що "OK"?)

"Кількість знайдених файлів збігається". (З чим?)

"Програма відмовляється виконувати команду". (Що означає «відмовляється»? Як це виглядає? Що має відбуватися?)

Опис дій як найменування модуля (підмодуля). Наприклад, "запуск програми" - це НЕ модуль або підмодуль. Модуль чи підмодуль — це деякі частини докладання, а не його поведінка. Порівняйте: "дихальна система" - це модуль людини, але "дихання" - ні.

Опис подій або процесів як кроки або очікувані результати. Наприклад, як крок сказано: «Введення спецсимволів у поле X». Це було б стерпною назвою тест-кейсу, але не годиться як крок, який повинен бути сформульований як "Ввести спецсимволи в поле X".

Набагато страшніше, якщо подібне зустрічається в очікуваних результатах. Наприклад, там написано: "Відображення швидкості читання в панелі X". І що? Воно має початися, продовжитися, завершитися, не починатися, якимось чином змінитись (наприклад, змінитись повинна розмірність даних), якось на щось вплинути? Тест-кейс стає абсолютно безглуздим, так як такий очікуваний результат неможливо порівняти з фактичною поведінкою програми.

«Вигадування» особливостей поведінки програми. Так, часто у вимогах відсутні самоочевидні речі, але нерідко трапляються й неякісні (наприклад, неповні) вимоги, які потрібно покращувати, а не «телепатично компенсувати».

Наприклад, у вимогах сказано, що «застосунок повинен відображати діалогове вікно збереження із зазначеним за умовчанням каталогом». Якщо з контексту (сусідних вимог, інших документів) нічого не вдається дізнатися про цей таємничий «каталог за умовчанням», потрібно поставити запитання. Не можна просто записати в очікуваних результатах «відображається діалогове вікно збереження із вказаним за замовчуванням каталогом» (як ми перевіримо, що обрано саме вказаний за замовчуванням каталог, а не якийсь інший?). І вже тим більше не можна в очікуваних результатах писати «відображається діалогове вікно збереження з обраним каталогом “C:/SavedDocuments”» (звідки взялося це «C:/SavedDocuments», - не ясно, тобто воно явно вигадане з голови і, швидше за все, вигадано неправильно).

Відсутність опису приготування до виконання тест-кейсу. Часто для коректного виконання тест-кейсу необхідно якимось особливим чином настроїти оточення. Припустимо, що ми перевіряємо програму, яка виконує резервне копіювання файлів. Якщо тест виглядає приблизно так, то виконує його співробітник буде збентежений, так як очікуваний результат здається просто маренням. Звідки взялося «~200»? Що це взагалі таке?

Кроки виконання	Очікувані результати
1. Натиснути на панелі «Головна» кнопку «Швидка дедублікація ». 2. Вибрати каталог "C: / MyData ".	1. Кнопка «Швидка дедублікація » переходить у втоплений стан і змінює колір із сірого на зелений. 2. На панелі «Стан» у полі «Дублікати» відображається «~200».

І зовсім інакше цей тест-кейс сприймався б, якби в приготуванні було сказано: «Створити каталог “C:/ MyData ” з довільним набором підкаталогів (глибина вкладеності не менше п'яти). В отриманому дереві каталогів розмістити 1000 файлів, з яких 200 мають однакові імена та розміри, але НЕ внутрішній вміст».

Повне дублювання (копіювання) одного і того ж тест-кейсу на рівнях димового тестування, тестування критичного шляху, розширеного тестування. Багато ідей природним чином розвиваються від рівня до рівня, але вони повинні саме розвиватися, а не дублюватися. Порівняйте:

	Димове тестування	Тестування критичного шляху	Розширене тестування
Погано	Запуск програми	Запуск програми.	Запуск програми.
Добре	Запуск програми.	Запуск програми з командного рядка. Запуск програми через ярлик на робочому столі. Запуск програми через меню "Пуск".	Запуск програми з командного рядка в активному режимі. Запуск програми з командного рядка у фоновому режимі. Запуск програми через ярлик на робочому столі від імені адміністратора. Запуск програми через меню «Пуск» зі списку програм, що часто запускаються.

Занадто довгий перелік кроків, що не належать до суті (мети) тест-кейсу. Наприклад, ми хочемо перевірити коректність одностороннього режиму друку з нашої програми на дуплексному принтері. Порівняйте:

Погано	Добре
Односторонній друк 1. Запустити програму. 2. У меню вибрати "Файл" -> "Відкрити". 3. Вибрати будь-який DOCX-файл, який складається з кількох сторінок. 4. Натисніть кнопку «Відкрити». 5. У меню вибрати "Файл" -> "Друк". 6. У списку «Двосторонній друк» виберіть пункт «Ні». 7. Натиснути кнопку "Друк". 8. Закрити файл. 9. Закрити програму.	**Односторонній друк** 1. Відкрити будь-який DOCX-файл, що містить три і більше непусті сторінки. 2. У діалоговому вікні «Налаштування друку» у списку «Двосторонній друк» виберіть «Ні». 3. Здійснити друк документа на принтері, який підтримує двосторонній друк.

Зліва ми бачимо величезну кількість дій, які не стосуються безпосередньо того, що перевіряє тест-кейс. Тим більше що запуск і закриття програми, відкриття файлу, робота меню та інше або будуть покриті іншими тест-кейсами (зі своїми відповідними цілями), або насправді є самоочевидними (адже логічно, що не можна відкрити застосунком файл, якщо програма не запущена) і не потребують опису кроків, які тільки створюють інформаційний шум і займають час на написання та прочитання.

Некоректне найменування елементів інтерфейсу чи його властивостей. Іноді з контексту зрозуміло, що автор тест-кейсу мав на увазі, але іноді це стає реальною проблемою. Наприклад, ми бачимо тест-кейс із назвою «Закриття програми кнопками "Close" та "Close window"». Вже тут виникає подив з приводу того, в чому ж відмінність цих кнопок, та й про що взагалі йдеться. Нижче (у кроках тест-кейсу) автор пояснює: «У робочій панелі внизу екрана натиснути "window"». Ага! Ясно. Але «Close window» - це НЕ кнопка, це пункт системного контекстного меню програми на панелі завдань.

Ще один чудовий приклад: «Вікно програми згорнеться у вікно меншого діаметра». Вікно кругле? Чи має стати круглим? А може, тут і зовсім мова про два різні вікна, і одне має опинитися всередині другого? Або, все ж таки «розмір вікна зменшується» (до речі, наскільки?). А його геометрична форма залишається прямокутною?

І, нарешті, приклад, через який можна написати звіт про дефект на цілком коректно працюючий застосунок: «У системному меню вибрати “Фіксація розташування”». Здавалося б, що тут поганого? Але потім з'ясовується, що мало на увазі головне меню програми, а не системне меню.

Нерозуміння принципів роботи програми та викликана цим некоректність тест-кейсів. Класикою жанру є закриття програми: той факт, що вікно програми «зникло» (сюрприз: наприклад, воно згорнулося в область повідомлення панелі завдань), або програма відключила інтерфейс користувача, продовживши функціонувати у фоновому режимі, зовсім не є ознакою того, що воно завершило роботу.

Перевірка типової «системної» функціональності. Якщо тільки ваша програма не написана з використанням якихось особливих бібліотек і технологій і не реалізує якусь нетипову поведінку, немає необхідності перевіряти системні кнопки, системні меню, згортання вікна і т.д. Імовірність зустріти тут помилку прагне нуля. Якщо все ж таки дуже хочеться, можна вписати ці перевірки як уточнення деяких дій на рівні тестування критичного шляху, але створювати для них окремі тест-кейси не потрібно.

Неправильна поведінка програми як очікуваний результат. Таке не допускається за визначенням. Не може бути тест-кейсу з кроком у стилі «поділити на нуль» з очікуваним результатом «крах програми з втратою даних». Очікувані результати завжди описують правильну поведінку програми - навіть у найстрашніших стресових тест-кейсах.

Загальна некоректність тест-кейсів. Може бути викликана безліччю причин і висловлюватися безліччю образів, але класичний приклад:

Кроки виконання	Очікувані результати
4. Закрити програму натисканням Alt+F4. 5. Вибрати в меню "Поточний стан".	4. Програма завершує роботу. 5. Відображається вікно із заголовком «Поточний стан» та вмістом, що відповідає малюнку 999.99.

Тут або не вказано, що виклик вікна «Поточний стан» відбувається десь в іншому застосунку, або залишається загадкою, як викликати це вікно у програми, що завершила роботу. Запустити заново? Можливо, але у тест-кейсі цього не сказано.

Неправильне розбиття наборів даних на класи еквівалентності. Справді, іноді класи еквівалентності може бути дуже неочевидними. Але помилки трапляються й у досить простих випадках. Припустимо, у вимогах сказано, що розмір файлу може бути від 10 до 100 КБ (включно). Розбиття за розміром 0-9 КБ, 10-100 КБ, 101 КБ помилково, так як кілобайт не є неподільною одиницею. Таке помилкове розбиття не враховує, наприклад, розміри 9.5 КБ, 100.1 КБ, 100.7 КБ і т.д. Тому

тут варто застосовувати нерівності: 0 КБ < розмір < 10 КБ, 10 КБ < розмір < 100 КБ, 100 КБ < розмір.

Тест-кейси, що не належать до тестованого застосунка . Наприклад, нам слід протестувати фотогалерею на сайті. Відповідно, наступні тест-кейси ніяк не відносяться до фотогалереї (вони тестують браузер, операційну систему користувача, файловий менеджер і т.д. - але не наша програма, ні його серверну, ні навіть клієнтську частину):

Файл із мережного диска.

Файл зі знімного носія.

Файл, заблокований іншою програмою.

Файл, відкритий іншим програмою.

Файл, до якого користувач не має прав доступу.

Вручну вказати шлях до файлу.

Файл із глибоко розташованої піддиректорії.

Формальні або суб'єктивні перевірки. Найчастіше цю помилку можна зустріти у пунктах чек-листа. Можливо, у автора в голові і був чіткий і докладний план, але з наступних прикладів неможливо зрозуміти, що буде зроблено з застосунком, і який результат ми повинні очікувати:

«Сконвертувати».

«Перевірити метод getMessage ( )».

«Некоректна робота у коректних умовах».

«Швидкість».

«Обсяг даних».

«Має працювати швидко».

В окремих виняткових ситуаціях можна заперечити, що з контексту та подальшої деталізації стає зрозуміло, що мало на увазі. Але найчастіше ніякого контексту і подальшої деталізації немає, тобто наведені приклади оформлені як окремі повноправні пункти чек-листа. Так не можна.