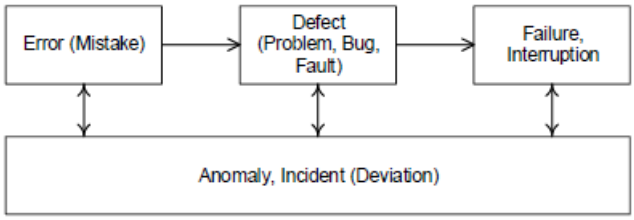
Лекція 4 Звіти про дефекти

4.1. Помилки, дефекти, збої, відмови.

Розберемося з широким спектром синонімів, якими позначають проблеми з програмними продуктами та іншими артефактами та процесами, що супроводжують їхню розробку.

У силабусі ISTQB написано, що людина робить помилки, які призводять до виникнення дефектів у коді, які, у свою черга, призводять до збоїв та відмов додатка (проте збої та відмови можуть виникати і через зовнішні умови, такі як електромагнітний вплив на обладнання тощо) .

Таким чином, можна побудувати схему в запозичення і проблем у розробці програмних продуктів (рис. 4.1).



Малюнок 4 . 1 - Взаємозв'язок проблем у розробці програмних продуктів

Розглянемо усі відповідні терміни.

Помилка - дія людини, що призводить до некоректних результатів .

Цей термін дуже часто використовують як найбільш універсальний, що описує будь-які проблеми («помилка людини», «помилка в коді», «помилка в документації», «помилка виконання операції», «помилка передачі даних», «помилковий результат» тощо) .) Більше того, куди частіше ви зможете почути «звіт про помилку», ніж «звіт про дефект». І це нормально, так склалося історично, до того ж термін «помилка» насправді дуже широкий.

Дефект – недолік у компоненті чи системі, здатний призвести до ситуації збою чи відмови .

Цей термін також розуміють досить широко, говорячи про дефекти в документації, налаштування, вхідні дані і т.д. Цей термін посередині

— безглуздо писати звіти про людські помилки, так само як і майже марно просто описувати прояви збоїв і відмов — потрібно докопатися до їх причини, і першим кроком у цьому напрямі є саме опис дефекту.

Збій — відмова, що самоусувається, або одноразова відмова, що усувається незначним втручанням оператора.

Відмова - подія, що полягає у порушенні працездатного стану об'єкта.

Ці терміни швидше стосуються теорії надійності і не часто зустрічаються в повсякденній роботі тестувальника, але саме збої та відмови є тим, що тестувальник помічає в процесі тестування (і відштовхуючись від чого проводить дослідження з метою виявити дефект і його причини).

Аномалія або інцидент - будь-яке відхилення (фактичного) стану, поведінки, значення, результату, властивості від очікувань спостерігача, сформованих на основі вимог, специфікацій, іншої документації або досвіду і здорового глузду.

Логічно випливає, що дефекти можуть зустрічатися не тільки в коді програми, а й у будь-якій документації, в архітектурі та дизайні, в налаштуваннях тестованого додатка або тестового оточення де завгодно.

4.2 . Звіт про дефект та його життєвий цикл

Як було зазначено, при виявленні дефекту тестувальник створює звіт про дефект.

Звіт про дефект — документ, що описує та пріоритизує виявлений дефект, а також сприяє його усуненню.

Як випливає із самого визначення, звіт про дефект пишеться з такими основними цілями:

· Надати інформацію про проблему - повідомити проектну команду та інших зацікавлених осіб про наявність проблеми, описати суть проблеми;

· пріоритизувати проблему - визначити ступінь небезпеки проблеми для проекту та бажані терміни її усунення;

· сприяти усуненню проблеми - якісний звіт про дефект не тільки надає всі необхідні подробиці для розуміння суті того, що

сталося, але також може містити аналіз причин виникнення проблеми та рекомендації щодо виправлення ситуації.

На останній цілі слід зупинитися докладніше. Є думка, що «добре написана звіт про дефект — половина вирішення проблеми для програміста». Як ми побачимо далі, від повноти, коректності, акуратності, подробиці та логічності звіту про дефект залежить дуже багато - одна і та ж проблема може бути описана так, що програмісту залишиться буквально виправити пару рядків коду, а може бути описана і так , що сам автор звіту наступного дня зможе зрозуміти, що він мав на увазі.

"Надціль" написання звіту про дефект полягає в швидкому виправленні помилки (а в ідеалі - і недопущення її виникнення в майбутньому). Тому якості звітів про дефект слід приділяти особливу, підвищену увагу.

Звіт про дефект (і сам дефект разом з ним) проходить певні стадії життєвого циклу, які схематично можна показати так (мал . 4 . 2):

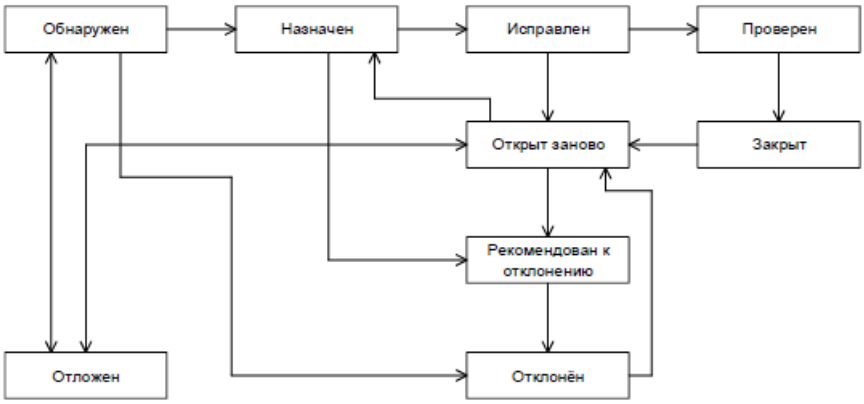


Рисунок 4. 2 - Життєвий цикл звіту про дефект з найбільш типовими переходами між станами

· Виявлено - початковий стан звіту (іноді називається "Новий"), в якому він знаходиться відразу після створення. Деякі засоби також дозволяють спочатку створювати чернетку (draft) і лише потім публікувати звіт.

· Призначений - у цей стан звіт переходить з моменту, коли хтось із проектної команди призначається відповідальним за виправлення дефекту. Призначення відповідального здійснюється або рішенням лідера команди розробки, або колегіально, або за

добровільним принципом, або іншим способом, прийнятим у команді, або виконується автоматично на основі певних правил.

· Виправлений - у цей стан звіт перекладає відповідальний за виправлення дефекту член команди після виконання відповідних дій з виправлення.

· Перевірений - у цей стан звіт перекладає тестувальник, який переконався, що дефект насправді був усунений. Як правило, таку перевірку виконує тестувальник, який спочатку написав звіт про дефект.

Щодо того, чи повинен перевіряти факт усунення дефекту саме той тестувальник, який його виявив, чи обов'язково інший, є багато «священних воєн». Прихильники другого варіанта стверджують, що новий погляд людини, раніше не знайомої з цим дефектом, дозволяє їй у процесі верифікації з великою ймовірністю виявити нові дефекти.

Незважаючи на те, що така точка зору має право на існування, все ж таки зазначимо: при грамотній організації процесу тестування пошук дефектів ефективно відбувається на відповідній стадії роботи, а верифікація силами тестувальника, який виявив цей дефект, все ж таки дозволяє суттєво заощадити час.

Набір стадій життєвого циклу, їх найменування та принцип переходу від стадії до стадії може різнитися в різних інструментальних засобах управління звітами про дефекти. Більше того, багато таких засобів дозволяють гнучко налаштовувати ці параметри. На рис . 4 . 2 показаний лише загальний принцип.

· Закритий - стан звіту, що означає, що за даним дефектом не планується ніяких подальших дій (хоча, звичайно, ніщо не заважає в майбутньому цьому дефекту стати «відновленим») . Тут є деякі розбіжності у життєвому циклі, прийнятому у різних інструментальних засобах управління звітами про дефекти .

У деяких засобах існують обидва стани — «Перевірено» та «Закрито», щоб підкреслити, що в стані «Перевірено» ще можуть знадобитися якісь додаткові дії (обговорення, додаткові перевірки у нових білдах і т.д.), у той час як стан «Закритий» означає «з дефектом покінчено, більше до цього питання не повертаємося».

У деяких засобах одного із станів немає (воно поглинається іншим).

У деяких засобах стан «Закритий» або «Відхилений» звіт про дефект може бути переведений з безлічі попередніх станів з резолюціями на кшталт:

▪ «Не є дефектом» — програма так і повинна працювати, описана поведінка не є аномальною.

▪ «Дублікат» — цей дефект вже описано в іншому звіті.

▪ «Не вдалося відтворити» — розробникам не вдалося відтворити проблему на своєму обладнанні.

▪ "Не буде виправлено" - дефект є, але з якихось серйозних причин його вирішено не виправляти.

▪ "Неможливо виправити" - непереборна причина дефекту знаходиться поза областю повноважень команди розробників, наприклад, існує проблема в операційній системі або апаратному забезпеченні, вплив якої усунути розумними способами неможливо.

Як було щойно підкреслено, у деяких засобах звіт про дефект у подібних випадках буде переведений у стан «Закритий», у деяких — у стан «Відхилений», у деяких частина випадків закріплена за станом «Закритий», частина — за «Відхилений» .

· Відкритий заново - у цей стан (як правило, зі стану «Виправлено») звіт перекладає тестувальник, який переконався, що дефект, як і раніше, відтворюється на білді, в якому він вже повинен бути виправлений.

· Рекомендований до відхилення - у цей стан звіт про дефект може бути переведений з багатьох інших станів з метою винести на розгляд питання про відхилення звіту з тієї чи іншої причини. Якщо рекомендація є обґрунтованою, звіт переводиться у стан «Відхилений».

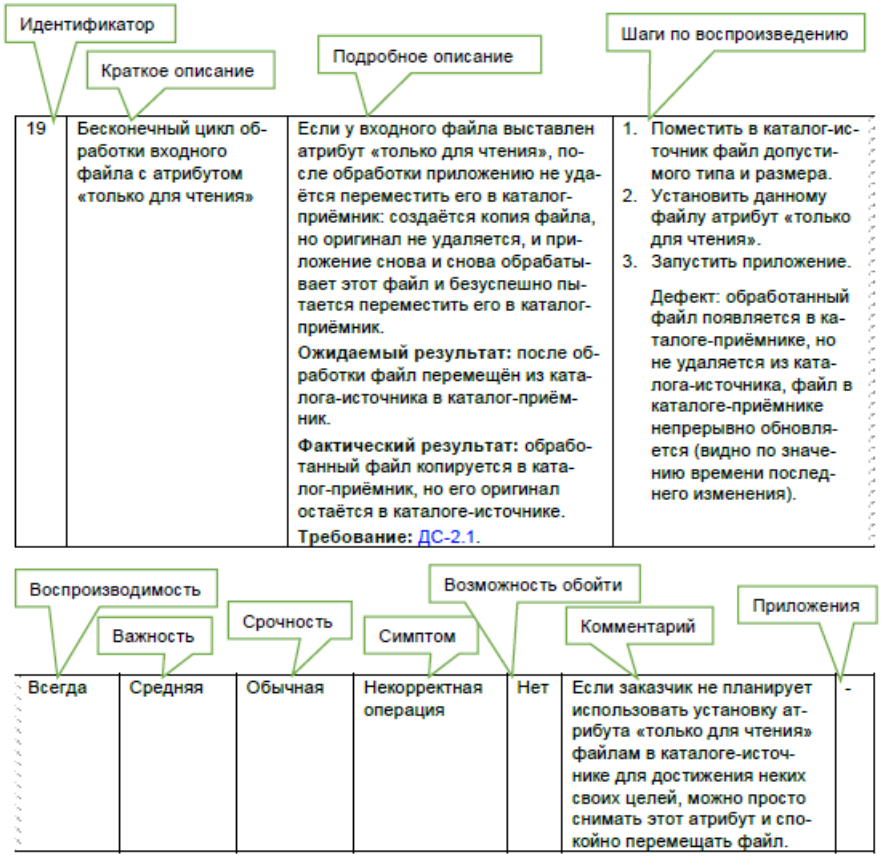
· Відхилений — у цей стан звіт перекладається у випадках, докладно описаних у пункті «Закритий», якщо засіб керування звітами про дефекти передбачає використання цього стану замість стану «Закритий» для тих чи інших резолюцій щодо звіту.

· Відкладений — у цей стан звіт перекладається у разі, якщо виправлення дефекту найближчим часом є нераціональним або не є можливим, проте є підстави вважати, що в найближчому майбутньому ситуація виправиться (вийде нова версія бібліотеки, повернеться з відпустки фахівець із якоїсь технології, зміняться вимоги замовника та ін.).

4.3 . Атрибути (поля) звіту про дефект

Залежно від інструментального засобу управління звітами про дефекти, зовнішній вигляд їх запису може трохи відрізнятися, можуть бути додані або прибрані окремі поля, але концепція залишається незмінною.

Загальний вигляд усієї структури звіту про дефект представлений на рис . 4 . 4 .



Малюнок 4 . 4 — Загальний вигляд звіту про дефект

Тепер розглянемо кожний атрибут докладно.

Ідентифікатор - Унікальне значення, що дозволяє однозначно відрізнити один звіт про дефект від іншого і використовуване в різних посиланнях. У загальному випадку ідентифікатор звіту про дефект може бути просто унікальним номером, але (якщо дозволяє інструментальний засіб управління звітами) може бути й куди складніше: включати префікси, суфікси та інші осмислені компоненти, що дозволяють швидко визначити суть дефекту та частину додатку (або вимог) , До якої він відноситься.

Короткий опис - Має у гранично лаконічній формі давати вичерпну відповідь на запитання «Що сталося?» "Де це сталося"? «За яких умов це сталося?». Наприклад: «Відсутній логотип на сторінці привітання, якщо користувач є адміністратором»:

· Що сталося? Немає логотипу.

· Де це сталося? На сторінці вітання.

· За яких умов це сталося? Якщо користувач є адміністратором.

Однією з найбільших проблем для тестувальників-початківців є саме заповнення поля «короткий опис», який одночасно повинен:

· Містити гранично коротку, але в той же час достатню для розуміння суті проблеми інформацію про дефект;

· відповідати на щойно згадані питання («що, де і за яких умов трапилося») або як мінімум на ті 1–2 питання, які застосовуються до конкретної ситуації;

· бути досить коротким, щоб повністю поміщатися на екрані (у тих системах управління звітами про дефекти, де кінець цього поля обрізається або призводить до появи скролінгу);

· При необхідності містити інформацію про оточення, під яким було виявлено дефект;

· по можливості не дублювати короткі описи інших дефектів (і навіть не бути схожими на них), щоб дефекти було складно переплутати чи вважати дублікатами один одного;

· бути закінченою пропозицією англійської (або іншої) мови, побудованою за відповідними правилами граматики.

Для створення хороших коротких описів дефектів рекомендується використовувати наступний алгоритм:

1. Повноцінно зрозуміти суть проблеми. Доки у тестувальника немає чіткого, кристально чистого розуміння того, «що зламалося», писати звіт про дефект взагалі навряд чи варто.

2. Сформулювати докладний опис дефекту - спочатку без огляду на довжину тексту, що вийшов.

3. Прибрати з детального опису все зайве, уточнити важливі деталі.

4. Виділити у докладному описі слова (словосполучення, фрагменти фраз), відповідальні питання, «що, де і яких умовах сталося».

5. Оформити отримане у пункті 4 у вигляді закінченої граматично правильної пропозиції.

6. Якщо пропозиція вийшла занадто довгою, переформулювати її, скоротивши довжину (за рахунок підбору синонімів, використання загальноприйнятих абревіатур та скорочень). До речі, в англійській мові пропозиція майже завжди буде коротшою за український аналог.

Розглянемо кілька прикладів застосування цього алгоритму.

Ситуація 1. Тестування піддається деякий веб-додаток, поле опису товару має допускати введення максимум 250 символів; у процесі тестування виявилося, що цього обмеження немає.

1. *Суть проблеми* : дослідження показало, що ні на клієнтській, ні на серверній частині немає жодних механізмів, які перевіряють або обмежують довжину введених у поле «Про товар» даних.

2. *Вихідний варіант докладного опису* : у клієнтській та серверній частині програми відсутні перевірка та обмеження довжини даних, що вводяться в полі «Про товар» на сторінці http://testapplication/admin/goods/edit/.

3. *Кінцевий варіант докладного опису* :

· Фактичний результат: в описі товару (поле «Про товар», http://testapplication/admin/goods/edit/) відсутні перевірка та обмеження довжини тексту, що вводиться (MAX=250 символів).

· Очікуваний результат: у разі спроби вводу 251+ символів виводиться повідомлення про помилку.

4. *Визначення «що, де і за яких умов сталося* »:

· Що: відсутні перевірка та обмеження довжини тексту, що вводиться.

· Де: опис товару, поле «Про товар», http://testapplication/admin/goods/edit/.

· За яких умов: – (у разі дефект присутній завжди, незалежно від будь-яких особливих умов).

5. *Первинне формулювання* : відсутні перевірка та обмеження максимальної довжини тексту, що вводиться в полі «Про товар» опису товару.

6. *Скорочення (підсумковий короткий опис*): немає обмеження максимальної довжини поля «Про товар». Англійський варіант: no check for "Про товар" max length.

Ситуація 2. Спроба відкрити в програмі порожній файл призводить до краху клієнтської частини програми та втрати незбережених даних користувача на сервері.

1. *Суть проблеми* : клієнтська частина програми починає «наосліп» читати заголовок файлу, не перевіряючи ні розмір, ні коректність формату, нічого; виникає певна внутрішня помилка, і клієнтська частина програми некоректно припиняє роботу, не закривши сесію із сервером; сервер закриває сесію по таймууту (повторний запуск клієнтської частини запускає нову сесію, так що стара сесія та всі дані в ній у будь-якому випадку втрачені).

2. *Вихідний варіант докладного опису* : некоректний аналіз файлу, що відкривається клієнтом, призводить до краху клієнта і незворотної втрати поточної сесії з сервером.

3. *Кінцевий варіант докладного опису* :

· Фактичний результат: відсутність перевірки коректності файлу (у тому числі порожнього), що відкривається клієнтською частиною, призводить до краху клієнтської частини і незворотної втрати поточної сесії з сервером.

· Очікуваний результат: проводиться аналіз структури файлу, що відкривається; у разі виявлення проблем з'являється повідомлення про неможливість відкриття файлу.

4. *Визначення «що, де і за яких умов сталося»:*

Що: крах клієнтської частини програми.

· Де: – (конкретне місце у додатку визначити навряд чи можливо).

· За яких умов: при відкритті порожнього або пошкодженого файлу.

5. *Первинне формулювання* : відсутність перевірки коректності файлу, що відкривається, призводить до краху клієнтської частини програми і втрати користувацьких даних.

6. *Скорочення (підсумковий короткий опис*): крах клієнта та втрата даних при відкритті пошкоджених файлів. Англійський варіант: client crash and data loss on damaged/empty files opening.

Ситуація 3. Вкрай рідко через абсолютно незрозумілі причини на сайті порушується відображення всього українського тексту (як

статичних написів, так і даних з бази даних, що генеруються динамічно тощо — все «стає питаннями»).

1. *Суть проблеми* : фреймворк, на якому збудовано сайт, підвантажує специфічні шрифти з віддаленого сервера; якщо з'єднання обривається, потрібні шрифти не підвантажуються, і використовуються стандартні шрифти, в яких немає українських символів.

2. *Вихідний варіант докладного описи* : помилка завантаження шрифтів з віддаленого сервера призводить до використання локальних несумісних із потрібним кодуванням шрифтів.

3. *Кінцевий варіант докладного опису* :

· Фактичний результат: періодична неможливість завантажити шрифти з віддаленого сервера призводить до використання локальних шрифтів, несумісних з необхідним кодуванням.

· Очікуваний результат: необхідні шрифти завжди підвантажуються (або використовується локальне джерело необхідних шрифтів).

4. *Визначення «що, де і за яких умов сталося* »:

· Що: використовуються несумісні з необхідним кодуванням шрифти.

· Де: - по всьому сайту.

· За яких умов: у разі помилки з'єднання з сервером, з якого підвантажуються шрифти.

5. *Первинне формулювання* : періодичні збої зовнішнього джерела шрифтів призводять до збою відображення російського тексту.

6. *Скорочення (підсумковий короткий опис*): неправильний показ українського тексту за недоступності зовнішніх шрифтів. Англійський варіант: wrong presentation of Ukraine text in case of external fonts inaccessibility.

Повертаємося до розгляду полів звіту про дефект.

Докладний опис – подає у розгорнутому вигляді необхідну інформацію про дефект, а також (обов'язково!) опис фактичного результату, очікуваного результату та посилання на вимогу (якщо це можливо).

Приклад детального опису:

Якщо адміністратор входить до системи, на сторінці привітання відсутній логотип.

Фактичний результат: логотип відсутній у лівому верхньому кутку сторінки .

Очікуваний результат: логотип відображається у верхньому лівому куті сторінки.

Вимога: R245.3.23b.

На відміну від короткого опису, який, як правило, є одним реченням, тут можна і потрібно давати докладну інформацію. Якщо одна й та сама проблема (викликана одним джерелом) проявляється в декількох місцях програми, можна в докладному описі перерахувати ці місця.

Кроки з відтворення - Описують дії, які необхідно виконати для відтворення дефекту. Це поле схоже на кроки тест-кейсу, за винятком однієї важливої відмінності: тут дії прописуються максимально докладно, із зазначенням конкретних значень, що вводяться, і найдрібніших деталей, тому що відсутність цієї інформації в складних випадках може призвести до неможливості відтворення дефекту.

Приклад кроків відтворення:

1. Відкрити http://testapplication/admin/login/.

2. Авторизуватися з ім'ям "defaultadmin" та паролем "dapassword".

Дефект: у лівому верхньому куті сторінки немає логотипу (замість нього відображається порожній простір з написом «logo»).

Відтворюваність показує, при кожному проходженні кроків відтворення дефекту вдається викликати його прояв. Це поле набирає всього два значення: завжди (always) або іноді (sometimes).

Можна сміливо сказати, що відтворюваність «іноді» означає, що тестувальник не знайшов справжню причину виникнення дефекту. Це призводить до серйозних додаткових труднощів у роботі з дефектом:

· Тестувальнику потрібно витратити багато часу на те, щоб переконатися в наявності дефекту (бо одноразовий збій у роботі програми міг бути викликаний величезною кількістю сторонніх причин).

· Розробнику теж потрібно витратити час, щоб досягти прояву дефекту та переконатися у його наявності. Після внесення виправлень у додаток розробник фактично повинен покладатися лише з власний професіоналізм, т.к. навіть багаторазове проходження кроками відтворення у разі не гарантує, що дефект було виправлено (можливо, ще через 10–20 повторень він проявився).

· Тестувальнику, що верифікує виправлення дефекту і зовсім залишається вірити розробнику на слово з тієї ж причини: навіть якщо він спробує відтворити дефект 100 разів і потім припинить спроби, може так статися, що на 101-й раз дефект все ж таки відтворився б.

Як легко здогадатися, така ситуація є вкрай неприємною, а тому рекомендується один раз витратити час на ретельну діагностику проблеми, знайти її причину та перевести дефект у розряд відтворюваних завжди.

Важливість – показує ступінь шкоди, яка завдається проекту існуванням дефекту.

У випадку виділяють такі градації важливості:

· Критична - існування дефекту призводить до масштабних наслідків катастрофічного характеру, наприклад: втрата даних, розкриття конфіденційної інформації, порушення ключової функціональності програми і т.д.

· Висока - існування дефекту приносить відчутні незручності багатьом користувачам у межах їх типової діяльності, наприклад: недоступність вставки з буфера обміну, непрацездатність загальноприйнятих клавіатурних комбінацій, необхідність перезапуску програми під час виконання типових сценаріїв роботи.

· Середня — існування дефекту слабко впливає на типові сценарії роботи користувачів, або існує обхідний шлях досягнення мети, наприклад: діалогове вікно не закривається автоматично після натискання кнопок «OK»/«Cancel», при роздрукуванні кількох документів поспіль не зберігається значення поля «Двосторонній друк» », переплутані напрями сортувань за певним полем таблиці.

· Низька - існування дефекту рідко виявляється незначним відсотком користувачів і майже не впливає на їх роботу, наприклад: друкарська помилка в глибоко вкладеному пункті меню налаштувань, якесь вікно відразу при відображенні розташоване незручно (потрібно перетягнути його в зручне місце), неточно відображається час до завершення операції копіювання файлів.

Терміновість - Вказує, як швидко дефект повинен бути усунений. У випадку виділяють такі градації терміновості:

· Найвища терміновість вказує на необхідність усунути дефект настільки швидко, наскільки це можливо. Залежно від контексту

«настільки швидко, наскільки можливо» може змінюватись від «у найближчому білді» до одиниць хвилин.

· Висока терміновість означає, що дефект слід виправити позачергово, т.к. його існування або вже об'єктивно заважає роботі, або почне створювати такі перешкоди у найближчому майбутньому.

· Звичайна терміновість означає, що дефект слід виправити в порядку загальної черги. Таке значення терміновості набуває більшість дефектів.

· Низька терміновість означає, що в найближчому майбутньому виправлення даного дефекту не вплине на підвищення якості продукту.

Декілька додаткових міркувань про важливість та терміновість варто розглянути окремо.

Одне з найчастіших питань відноситься до того, який між ними зв'язок. Ніякий. Для кращого розуміння цього факту можна порівняти важливість та терміновість з координатами X та Y точки на площині. Хоча «на побутовому рівні» і здається, що дефект із високою важливістю слід виправити насамперед, насправді ситуація може виглядати зовсім інакше.

Щоб проілюструвати цю думку докладніше, повернемося до переліку градацій: чи ви помітили, що для різних ступенів важливості приклади наведені, а для різних ступенів терміновості — ні? І це невипадково.

Знаючи суть проекту та суть дефекту, його важливість визначити досить легко, т.к. ми можемо простежити вплив дефекту критерії якості, ступінь виконання вимог тієї чи іншої важливості тощо. Але терміновість виправлення дефекту можна визначити лише у конкретній ситуації.

Пояснимо на прикладі життя: наскільки для життя людини важлива вода? Дуже важлива, без води людина вмирає. Значить важливість води для людини можна оцінити як критичну. Але чи можемо ми відповісти на запитання «Як швидко людині потрібно випити води?» знаючи, про який ситуації йдеться? Якщо людина, яка розглядається, помирає від спраги в пустелі, терміновість буде найвищою. Якщо він просто сидить в офісі і думає, чи не попити чаю, терміновість буде звичайною або навіть низькою.

Симптом - дозволяє класифікувати дефекти щодо їх типового прояву. Не існує загальноприйнятого списку симптомів. Більше того,

далеко не в кожному інструментальному засобі управління звітами про дефекти є таке поле, а там, де воно є, його можна налаштувати. Як приклад розглянемо такі значення симптомів дефекту.

· Косметичний дефект – візуально помітний недолік інтерфейсу, що не впливає на функціональність програми (наприклад, напис на кнопці виконаний шрифтом не тієї гарнітури).

· Пошкодження/втрата даних - внаслідок виникнення дефекту спотворюються, знищуються (або не зберігаються) деякі дані (наприклад, при копіюванні файлів копії виявляються пошкодженими).

· Проблема в документації - дефект відноситься не до додатку, а до документації (наприклад, немає розділу посібника з експлуатації).

· Некоректна операція - деяка операція виконується некоректно (наприклад, калькулятор показує відповідь 17 при множенні 2 на 3).

· Проблема інсталяції - дефект проявляється на стадії встановлення та/або конфігурування програми.

· Помилка локалізації — щось у програмі не перекладено або перекладено неправильно на вибрану мову інтерфейсу.

· Нереалізована функціональність — якась функція програми не виконується або не може бути викликана (наприклад, у списку форматів для експорту документа немає кількох пунктів, які там мають бути).

· Проблема масштабованості - при збільшенні кількості доступних додатку ресурсів не відбувається очікуваного приросту продуктивності програми .

· Низька продуктивність - виконання деяких операцій займає неприпустимо багато часу.

· Крах системи – програма припиняє роботу або втрачає здатність виконувати свої ключові функції (також може супроводжуватися крахом операційної системи, веб-сервера тощо).

· Несподівана поведінка - у процесі виконання деякої типової операції додаток поводиться незвичайним (відмінним від загальноприйнятого) чином (наприклад, після додавання до списку нового запису активним стає не новий запис, а перший у списку).

· Недружня поведінка - поведінка програми створює користувачеві незручності в роботі (наприклад, на різних діалогових вікнах в різному порядку розташовані кнопки OK і Cancel).

· Розбіжність з вимогами - цей симптом вказують, якщо дефект складно співвіднести з іншими симптомами, проте додаток поводиться не так, як описано в вимогах.

Пропозиція щодо покращення — у багатьох інструментальних засобах управління звітами про дефекти для цього випадку є окремий вид звіту, т.к. пропозицію щодо поліпшення формально не можна вважати дефектом: додаток поводиться згідно з вимогами, але у тестувальника є обґрунтована думка про те, як ту чи іншу функціональність можна покращити.

Часто зустрічається питання, чи може в одного дефекту бути відразу кілька симптомів. Да може. Наприклад, крах системи часто веде до втрати або пошкодження даних. Але у більшості інструментальних засобів управління звітами про дефектах значення поля «Симптом» вибирається з списку, тому немає можливості вказати два і більше симптоми одного дефекту. У такій ситуації рекомендується вибирати або симптом, який найкраще описує суть ситуації , або «найнебезпечніший» симптом (наприклад, недружня поведінка, яка полягає в тому, що додаток не просить підтвердження перезапису існуючого файлу, призводить до втрати даних; тут «втрата даних» набагато доречніше, ніж «недружня поведінка»).

Можливість обійти — показує, чи є альтернативна послідовність дій, виконання якої дозволило б користувачеві досягти поставленої мети (наприклад, клавіатурна комбінація Ctrl+P не працює, але роздрукувати документ можна, вибравши відповідні пункти меню). У деяких інструментальних засобах керування звітами про дефекти це поле може просто приймати значення «Так» та «Ні», у деяких при виборі «Так» з'являється можливість описати обхідний шлях. Традиційно вважається, що дефекти без можливості обходу варто підвищити терміновість виправлення.

Коментар – може містити будь-які корисні для розуміння та виправлення дефекту дані. Іншими словами, сюди можна писати все те, що не можна писати до інших полів.

Вкладення - представляє собою не стільки поле, скільки список прикріплених до звіту про дефект додатків (копій екрана, що викликають збій файлів тощо).

Загальні рекомендації щодо формування додатків такі:

· Якщо ви сумніваєтеся, робити чи не робити програму, краще зробіть.

· Обов'язково додайте т.зв. «проблемні артефакти» (наприклад, файли, які додаток обробляє некоректно).

· Якщо ви додаєте копію екрана:

- Найчастіше вам буде потрібна копія активного вікна (Alt+PrintScreen), а не лише екрану (PrintScreen).
- Обріжте все зайве (використовуйте Snipping Tool або Paint у Windows, Pinta або XPaint у Linux).
- Позначте на копії екрана проблемні місця (обведіть, намалюйте стрілку, додайте напис - зробіть все необхідне, щоб проблема була помітна і зрозуміла з першого погляду).
- У деяких випадках варто зробити одне велике зображення з кількох копій екрана (розмістивши послідовно), щоб показати процес відтворення дефекту. Альтернативою цього рішення є створення кількох копій екрана, названих так, щоб імена утворювали послідовність, наприклад: br_9_sc_01.png, br_9_sc_02.png, br_9_sc_03.png.
- Збережіть копію екрану у форматі JPG (якщо важлива економія обсягу даних) або PNG (якщо важлива точна передача зображення без спотворень).

· Якщо ви додаєте відеоролик із записом того, що відбувається на екрані, обов'язково залишайте тільки той фрагмент, який відноситься до описуваного дефекту (це буде буквально кілька секунд або хвилин з можливих багатьох годин запису). Намагайтеся підібрати параметри кодеків так, щоб отримати мінімальний розмір ролика при збереженні достатньої якості зображення.

· Поекспериментуйте з різними інструментами створення копій екрана та запису відео з тим, що відбувається на екрані. Виберіть найбільш зручне для вас програмне забезпечення та привчіть себе постійно його використовувати.

4.4 . Інструментальні засоби керування звітами про дефекти

Так звані «інструментальні засоби управління звітами про дефекти» у звичайній розмовній мові називають «баг-трекінговими системами», «баг-трекерами» тощо. Але ми дотримуватимемося суворішої термінології.

Інструментальних засобів управління звітами про дефекти дуже багато , до того ж багато компаній розробляють свої внутрішні засоби вирішення цього завдання. Найчастіше такі інструментальні засоби є частинами інструментальних засобів управління тестуванням .

Як і у випадку з інструментальними засобами керування тестуванням, тут немає сенсу заучувати, як працювати зі звітами про дефекти в тому чи іншому засобі. Ми лише розглянемо загальний набір функцій, які зазвичай реалізуються такими засобами:

1. Створення звітів про дефекти, управління їх життєвим циклом з урахуванням контролю версій, прав доступу та дозволених переходів зі стану.
2. Збір, аналіз та надання статистики у зручній для сприйняття людиною формі.
3. Розсилка повідомлень, нагадувань та інших артефактів відповідним співробітникам.
4. Організація взаємозв'язків між звітами про дефекти, тест-кейсами, вимогами та аналіз таких зв'язків із можливістю формування рекомендацій.
5. Підготовка інформації для включення до звіту про результати тестування.
6. Інтеграція із системами управління проектами.

Іншими словами, гарний інструментальний засіб управління життєвим циклом звітів про дефекти не тільки позбавляє людину необхідності уважно виконувати велику кількість рутинних операцій, а й надає додаткові можливості, що полегшують роботу тестувальника і роблять її більш ефективною.

Для загального розвитку та кращого закріплення теми про оформлення звітів про дефекти ми розглянемо зараз кілька картинок з формами з різних інструментальних засобів.

Ми цілком свідомо не наводимо ніякого порівняння або докладного опису - подібних оглядів достатньо в Інтернеті, і вони стрімко застарівають у міру виходу нових версій продуктів, що оглядаються .

Але інтерес представляють окремі особливості інтерфейсу, куди ми звернемо увагу кожному з прикладів .

Jira (Рис. 4.4)

Малюнок 4 . 4 — Створення звіту про дефект JIRA

1. Project (проект) дозволяє вказати, якого проекту належить дефект.

2. Issue type (тип запису/артефакту) дозволяє вказати, що саме являє собою створюваний артефакт. *JIRA* дозволяє створювати не тільки звіти про дефекти, а й безліч інших артефактів, типи яких можна налаштовувати. За замовчуванням наведено:

· Improvement (пропозиція щодо покращення) - було детально розглянуто в розділі, присвяченому полям звіту про дефект.

· New feature (нова особливість) - опис нової функціональності, нової властивості, нової особливості продукту.

· Task (завдання) - якесь завдання для виконання тим чи іншим учасником проектної команди.

· Custom issue (довільний артефакт) - як правило, це значення при налаштуванні *JIRA* видаляють, замінюючи своїми варіантами, або перейменовують Issue.

3. Summary (короткий опис) дозволяє вказати короткий опис дефекту.

4. Priority (Терміновість) дозволяє вказати терміновість виправлення дефекту. за за замовчуванням *JIRA* пропонує такі варіанти:

- Highest (найвища терміновість).
- High (висока терміновість).
- Medium (звичайна терміновість).
- Low (низька терміновість).
- Lowest (найнижча терміновість).

Зауважте: за замовчуванням поля severity (важливість) немає. Але його можна додати.

5. Components (компоненти) містить перелік компонентів програми, порушених дефектом (хоча іноді тут перераховують симптоми дефектів).

6. Affected versions (зачеплені версії) містить перелік версій продукту, де проявляється дефект.

7. Environment (оточення) містить опис апаратної та програмної конфігурації, в якій проявляється дефект.

8. Description (Докладний опис) дозволяє вказати докладний опис дефекту.

9. Original estimate (початкова оцінка часу виправлення) дозволяє вказати початкову оцінку того, скільки часу займе усунення дефекту.

10. Remaining estimate (розрахунковий залишковий час виправлення) показує скільки часу залишилося від початкової оцінки.

11. Story points (оціночні одиниці) дозволяє вказати складність дефекту (або іншого артефакту) у спеціальних оцінних одиницях, прийнятих у гнучких методологіях управління проектами.

12. Labels (мітки) містить мітки (теги, ключові слова), за якими можна групувати та класифікувати дефекти та інші артефакти.

13. Epic/Theme (історія/область) містить перелік високорівневих міток, що описують великі області вимог, що відносяться до дефекту, великі модулі програми, великі частини предметної області, об'ємні користувальницькі історії і т.д.

14. External issue id (ідентифікатор зовнішнього артефакту) дозволяє пов'язати звіт про дефект або інший артефакт із зовнішнім документом.

15. Epic link (посилання на історію/область) містить посилання на історію/область, найближче до дефекту.

16. Has a story/s (історії) містить посилання та/або опис історій користувача, пов'язаних з дефектом (як правило, тут наводяться посилання на зовнішні документи).

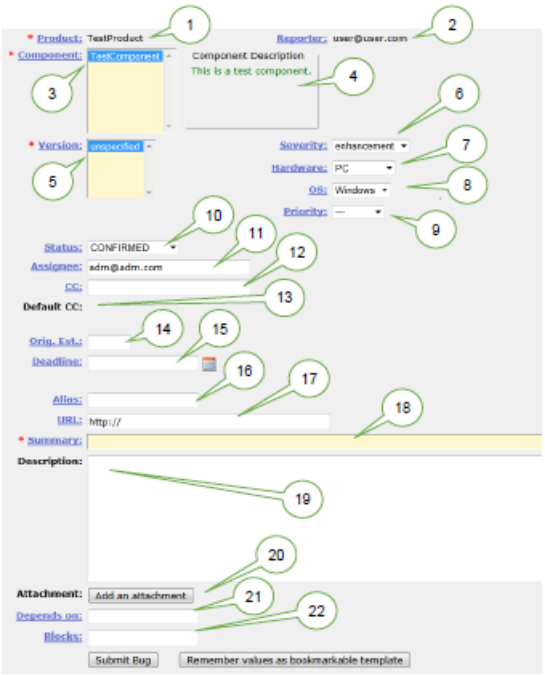
17. Tester (тестувальник) містить ім'я автора опису дефекту.

18. Additional information (додаткова інформація) містить додаткову інформацію про дефект.

19. Sprint (Спринт) містить номер спринту (2–4-тижнева ітерація розробки проекту в термінології гнучких методологій управління проектами), під час якого було виявлено дефект.

Багато додаткових полів і можливостей стають доступними при інших операціях з дефектами (переглядом або редагуванням створеного дефекту, перегляд звітів і т.д.).

Bugzilla (Рис. 4.5)



Малюнок 4 .5 — Створення звіту про дефект Bugzilla

1. Product (продукт) дозволяє вказати, якого продукту (проекту) належить дефект.

2. Reporter (автор звіту) містить e-mail автора опису дефекту.

3. Component (компонент) містить вказівку компонента програми, до якого відноситься дефект, що описується.

4. Component description (опис компонента) містить опис компонента програми, до якого відноситься дефект, що описується. Ця інформація завантажується автоматично під час вибору компонента.

5. Version (версія) містить вказівку версії продукту, де було виявлено дефект.

6. Severity (важливість) містить вказівку на важливість дефекту. За замовчуванням запропоновано такі варіанти:

- Blocker (блокуючий дефект) - дефект не дозволяє вирішити за допомогою програми певне завдання.
- Critical (критична важливість).
- Major (висока важливість).
- Normal (звичайна важливість).
- Minor (низька важливість).
- Trivial (найнижча важливість).
- Enhancement (пропозиція щодо покращення) – було описано докладно в розділі, присвяченому полям звіту про дефект.

7. Hardware (апаратне забезпечення) дозволяє вибрати профіль апаратного оточення, в якому проявляється дефект.

8. OS (операційна система) дозволяє вказати операційну систему, під якою проявляється дефект.

9. Priority (Терміновість) дозволяє вказати терміновість виправлення дефекту. За замовчуванням Bugzilla пропонує такі варіанти:

- Highest (найвища терміновість).
- High (висока терміновість).
- Normal (звичайна терміновість).
- Low (низька терміновість).
- Lowest (найнижча терміновість).

10. Status (Статус) дозволяє встановити статус звіту про дефект. За замовчуванням Bugzilla пропонує такі варіанти статусів:

- Unconfirmed (не підтверджено) - дефект поки не вивчений, і немає гарантії того, що він справді коректно описаний.
- Confirmed (підтверджено) – дефект вивчений, коректність опису підтверджена.
- In progress (в роботі) - ведеться робота з вивчення та усунення дефекту.

В офіційній документації рекомендується відразу після встановлення Bugzilla налаштувати набір статусів та правила життєвого циклу звіту про дефекти відповідно до прийнятих у вашій компанії правил.

11. Assignee (відповідальний) вказує e-mail учасника проектної команди, відповідального за вивчення та виправлення дефекту.

12. CC (повідомляти) містить список e-mail адрес учасників проектної команди, які будуть отримувати повідомлення про те, що відбувається з цим дефектом.

13. Default CC (повідомляти за замовчуванням) містить e-mail адресу(и) учасників проектної команди, які за умовчанням отримуватимуть повідомлення про те, що відбувається з будь-якими дефектами (найчастіше тут вказуються e-mail адреси розсилок).

14. Original estimation (початкова оцінка) дозволяє вказати початкову оцінку того, скільки часу займе усунення дефекту.

15. Deadline (крайній термін) дозволяє вказати дату, до якої дефект потрібно виправити.

16. Alias (псевдонім) дозволяє вказати коротку назву дефекту (можливо, у вигляді якоїсь абревіатури) для зручності згадування дефекту в різноманітних документах.

17. URL дозволяє вказати URL, де проявляється дефект (особливо актуально для веб-додатків).

18. Summary (короткий опис) дозволяє вказати на короткий опис дефекту.

19.Description (Докладний опис) дозволяє вказати докладний опис дефекту.

20. Attachment (додавання) дозволяє додати до звіту про дефект вкладення у вигляді прикріплених файлів.

21. Depends on (залежить від) дозволяє вказати перелік дефектів, які мають бути усунені до початку роботи з цим дефектом.

22. Blocks (блокує) дозволяє вказати перелік дефектів, до роботи з якими можна буде приступити тільки після усунення даного дефекту.

4.5 . Властивості якісних звітів про дефекти

Звіт про дефект може виявитися не якісним (а отже, ймовірність виправлення дефекту знизиться), якщо в ньому порушено одну з таких властивостей.

Ретельне заповнення всіх полів точною та коректною інформацією . Порушення цієї властивості відбувається з багатьох причин: недостатній досвід тестувальника, неуважність, лінь і т.д. Найяскравішими проявами такої проблеми вважатимуться такі:

· Частина важливих розуміння проблеми полів не заповнена. В результаті звіт перетворюється на набір уривчастих відомостей, використовувати які для виправлення дефекту неможливо.

· Наданої інформації недостатньо для розуміння суті проблеми. Наприклад, з такого поганого детального опису взагалі не ясно, про що йдеться: «Додаток іноді неправильно конвертує деякі файли».

· Надана інформація є некоректною (наприклад, вказані неправильні повідомлення, неправильні технічні терміни тощо). Найчастіше таке відбувається через неуважність (наслідки помилкового copy-paste та відсутності фінальної вичитки звіту перед публікацією).

· «Дефект» (саме так, у лапках) знайдений у функціональності, яка ще не була оголошена як готова до тестування. Тобто тестувальник констатує, що неправильно працює те, що й не мало (поки що!) правильно працювати.

· У звіті присутня жаргона лексика: як у прямому сенсі - не літературні висловлювання, і деякі технічні жаргонізми, зрозумілі вкрай обмеженому колу людей. Наприклад: "Фігово підчепилися чартники". (Малося на увазі: "Не всі таблиці кодувань завантажені успішно".)

· Звіт замість опису проблеми із додатком критикує роботу когось із учасників проектної команди. Наприклад: "Ну яким дурнем треба бути, щоб таке зробити?!"

· У звіті втрачена якась незначна на перший погляд, але за фактом критична для відтворення дефекту проблема. Найчастіше це проявляється у вигляді пропуску якогось кроку з відтворення, відсутності або недостатньої подробиці опису оточення, надмірно узагальненому вказівці значень, що вводяться і т.п.

· Звіту виставлені невірні (зазвичай, занижені) важливість чи терміновість. Щоб уникнути цієї проблеми, варто ретельно дослідити

дефект, визначати його найнебезпечніші наслідки та аргументовано відстоювати свою точку зору, якщо колеги вважають інакше.

· До звіту не додані необхідні копії екрана (особливо важливі для косметичних дефектів) або інші файли. Класика такої помилки: звіт описує неправильну роботу програми з деяким файлом, але сам файл не доданий.

· Звіт написаний безграмотно з погляду людської мови. Іноді на це можна заплющити очі, але іноді це стає реальною проблемою, наприклад: « На клавіатурі немає параметрів значення » (це реальна цитата; і сам автор так і не зміг пояснити, що ж мало на увазі).

Правильна технічна мова . Ця властивість однаково стосується і вимог, і тест-кейсів, і звітів про дефекти — будь-якої документації, а тому не будемо повторюватися .

Порівняйте два докладні описи дефекту:

Поганий докладний опис	Хороший докладний опис
Коли ми ніби хочемо прибрати папку, де щось усередині є, воно не запитує, чи ми хочемо.	Не здійснено запит підтвердження при видаленні непустого підкаталогу в каталозі SOURCE_DIR. Act: проводиться видалення не порожнього підкаталогу (з усім його вмістом) у каталозі SOURCE_DIR без запиту підтвердження. Exp: якщо у каталозі SOURCE_DIR додаток виявляє не порожній підкаталог, воно припиняє роботу з виведенням повідомлення “Non-empty subfolder [ім'я підкаталогу] in SOURCE_DIR folder detected. Remove it manu-ally або restart application with --force_file_opera-tions key to remove automatically.»

Специфічність опису кроків . Говорячи про тест-кейси, ми підкреслювали, що в їхніх кроках варто притримуватися золотої середини між специфічністю та спільністю. У звітах про дефекти перевагу, як правило, надають специфічності з дуже простої причини: брак якоїсь дрібної деталі може призвести до неможливості відтворення

дефекту. Тому, якщо у вас є хоч найменший сумнів, чи важлива якась деталь, вважайте, що вона важлива.

Порівняйте два набори кроків щодо відтворення дефекту:

Недостатньо специфічні кроки	Досить специфічні кроки
1. Надіслати на конвертацію файл допустимого формату та розміру, в якому український текст представлений у різних кодуваннях. Дефект: конвертація кодувань неправильна.	1. Надіслати на конвертацію файл у форматі HTML розміром від 100 КБ до 1 МБ, в якому український текст представлений у кодуваннях UTF8 (10 рядків по 100 символів) та WIN-1251 (20 рядків по 100 символів). Дефект: текст, який був представлений в UTF8, пошкоджений (представлений набором символів, що не читається).

У першому випадку відтворити дефект майже неможливо, т.к. він полягає в особливостях роботи зовнішніх бібліотек щодо визначення кодувань тексту в документі, у той час як у другому випадку даних достатньо якщо і не для розуміння суті того, що відбувається (дефект насправді дуже «хитрий»), то хоча б для гарантованого відтворення та визнання факту його наявності.

Ще раз головна думка: на відміну від тест-кейсу звіт про дефект може мати підвищену специфічність, і це буде меншою проблемою, ніж неможливість відтворення дефекту через надмірно узагальнений опис проблеми.

Відсутність зайвих дій чи їх довгих описів . Найчастіше ця властивість передбачає, що не потрібно в кроках відтворення дефекту 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 (особливо звернути увагу, щоб SOURCE_DIR та DESTINATION_DIR не збігалися та не були вкладені одна в одну у будь-якому поєднанні). Дефект: програма припиняє роботу з повідомленням "SOURCE_DIR and

Дефект: програма використовує перший параметр командного рядка як шлях до папки з вихідними файлами, і як шлях до папки з кінцевими файлами.	DESTINATION_DIR не може бути!" (зважаючи на все, перший параметр командного рядка використовується для ініціалізації імен обох каталогів.)

Друга за частотою помилка — початок кожного звіту про дефект із запуску програми та докладного опису щодо приведення його в той чи інший стан. Цілком допустимою практикою є написання у звіті про дефект приготувань (за аналогією з тест-кейсами) або опис потрібного стану програми в одному (першому) кроці. Порівняйте:

Погано	добре
1. Запустити програму з усіма правильними параметрами. 2. Зачекати більше 30 хвилин. 3. Надіслати на конвертацію файл допустимого формату та розміру. Дефект: додаток не обробляє файл.	Передмова: програма запущена та пропрацювала більше 30 хвилин. 1. Надіслати на конвертацію файл допустимого формату та розміру. Дефект: додаток не обробляє файл.

Відсутність дублікатів . Коли в проектній команді працює велика кількість тестувальників, може виникнути ситуація, за якої той самий дефект буде описаний кілька разів різними людьми. А іноді буває так, що навіть той самий тестувальник уже забув, що колись давно вже виявляв якусь проблему, і тепер описує її наново. Уникнути такої ситуації дозволяє наступний набір рекомендацій:

· Якщо ви не впевнені, що дефект не був описаний раніше, зробіть пошук за допомогою інструментального засобу управління життєвим циклом звітів про дефекти.

· Пишіть максимально інформативні короткі описи (бо пошук в першу чергу проводять за ними). Якщо на вашому проекті накопичиться безліч дефектів з короткими описами в стилі «Кнопка не працює», ви витратите дуже багато часу, щоразу перебираючи десятки таких звітів у пошуках потрібної інформації.

· Використовуйте максимум можливості вашого інструментального засобу: вказуйте у звіті про дефект компоненти програми, посилання на вимоги, розставляйте теги і т.д. — все це дозволить легко та швидко звузити у майбутньому коло пошуку.

· Вкажіть у докладному описі дефекту текст повідомлень від програми, якщо це можливо. За таким текстом можна знайти навіть той звіт, у якому решта інформації наведена у занадто загальному вигляді.

· Намагайтеся по можливості брати участь у т.зв. "зборах з прояснення", т.к. нехай ви і не запам'ятаєте кожен дефект або кожну історію користувача дослівно, але в потрібний момент може виникнути відчуття «щось таке я вже чув», яке змусить вас зробити пошук і підкаже, що саме шукати.

· Якщо ви виявили якусь додаткову інформацію, внесіть її до існуючого звіту про дефект (або попросіть зробити це його автора), але не створюйте окремий звіт.

Очевидність та зрозумілість . Описуйте дефект так, щоб у вашого звіту не виникло жодного сумніву в тому, що це дійсно дефект. Найкраще це властивість досягається рахунок ретельного пояснення фактичного і очікуваного результату, і навіть вказівки посилання вимогу у полі «Детальний опис».

Порівняйте:

Поганий докладний опис	Хороший докладний опис
Програма не повідомляє про виявлені підкаталоги в каталозі SOURCE_DIR.	Програма не повідомляє користувача про виявлені в каталозі SOURCE_DIR підкаталоги, що призводить до необґрунтованих очікувань користувачами обробки файлів у таких підкаталогах. Act: програма починає (продовжує) роботу, якщо в каталозі SOURCE_DIR знаходяться підкаталоги. Exp: якщо у каталозі SOURCE_DIR програма при запуску або в процесі роботи виявляє порожній підкаталог, вона автоматично видаляє (чи логічно це?), якщо а виявлено непустий підкаталог, додаток припиняє роботу з виведенням повідомлення «Non-empty subfolder [ім'я підкаталогу] in SOURCE_DIR folder detected. Remove it manu-ally або restart application with --force_file_opera-tions key to remove automatically.»

У першому випадку після прочитання докладного опису дуже хочеться запитати: І що? А воно хіба має повідомляти?»

Другий варіант докладного опису дає чітке пояснення, що така поведінка є помилковою згідно з поточним варіантом вимог. Більше того, у другому варіанті відмічено питання (а в ідеалі потрібно зробити відповідну позначку і в самій вимогі), що закликає переглянути алгоритм коректної поведінки додатка у подібній ситуації: тобто. ми не тільки якісно описуємо поточну проблему, але й порушуємо питання щодо подальшого поліпшення програми.

Простежуваність . З інформації, що міститься в якісному звіті, має бути зрозуміло, яку частину програми, які функції і які вимоги зачіпає дефект. Найкраще ця властивість досягається правильним використанням можливостей інструментального засобу управління звітами про дефекти: вказуйте у звіті про дефект компоненти додатка, посилання на вимоги, тест-кейси, суміжні звіти про дефекти (схожі дефекти; залежні та залежать від даного дефекти), розставляйте теги та і т.д.

Деякі інструментальні засоби дозволяють будувати візуальні схеми з урахуванням таких даних, що дозволяє управління простежуваністю навіть у дуже великих проектах перетворити з непосильної людини завдання на тривіальну роботу.

Окремі звіти для кожного нового дефекту . Існує два непорушні правила:

1. У кожному звіті описується рівно один дефект (якщо той самий дефект проявляється у кількох місцях, ці прояви перераховуються у докладному описі).
2. При виявленні нового дефекту створюється новий звіт. Не можна для опису нового дефекту правити старі звіти, переводячи їх у стан «відновлено».

Порушення першого правила призводить до об'єктивної плутанини, яку найпростіше проілюструвати так: уявіть, що в одному звіті описано два дефекти, один з яких був виправлений, а другий — ні. У який стан перекладати звіт? Невідомо.

Порушення другого правила взагалі породжує хаос: мало того що втрачається інформація про дефекти, що раніше виникали, так до того ж виникають проблеми з всілякими метриками і банальним здоровим глуздом. Саме щоб уникнути цієї проблеми на багатьох проектах правом переведення звіту зі стану «закритий» у стан «відкрито заново» має обмежене коло учасників команди.

Відповідність прийнятим шаблонам оформлення та традиціям . Як і у випадку з тест-кейсами, з шаблонами оформлення звітів про дефекти проблем не виникає: вони визначені зразком або екранною формою інструментального засобу управління життєвим циклом звітів про дефекти. Але щодо традицій, які можуть відрізнятися навіть у різних командах в рамках однієї компанії, то єдина порада: почитайте вже готові звіти про дефекти, перед тим як писати свої. Це може заощадити вам багато часу та сил.

4.6 . Логіка створення ефективних звітів про дефекти

При створенні звіту про дефект рекомендується дотримуватися наступного алгоритму:

0. Виявити дефект.

1. Зрозуміти суть проблеми.

2. Відтворити дефект.

3. Перевірити наявність опису знайденого вами дефекту у системі управління дефектами.

4. Сформулювати суть проблеми у вигляді «що зробили, що отримали, що очікували отримати».

5. Заповнити поля звіту, починаючи з детального опису.

6. Після заповнення всіх полів уважно перечитати звіт, виправивши неточності та додавши подробиці.

7. Ще раз перечитати звіт, т.к. у пункті 6 ви точно щось пропустили.

Тепер про кожен крок докладніше.

Зрозуміти суть проблеми . Все починається саме з розуміння того, що відбувається з додатком. Тільки за наявності такого розуміння ви зможете написати по-справжньому якісний звіт про дефект, правильно визначити важливість дефекту та дати корисні рекомендації щодо його усунення. В ідеалі звіт про дефект описує суть проблеми, а не її зовнішній прояв.

Порівняйте два звіти про ту саму ситуацію (додаток «Конвертер файлів» не розрізняє файли та символічні посилання на файли, що призводить до серії аномалій у роботі з файловою системою).

Поганий звіт, при написанні якого суть проблеми не зрозуміла:

Короткий опис	Докладний опис	Кроки з відтворення

1	2	3
Обробляються файли поза SOURCE_DIR.	Іноді з незрозумілих причин програма обробляє випадкові файли поза каталогом SOURCE_DIR. Act: обробляються окремі файли поза SOURCE_DIR. Exp: обробляються лише файли, що знаходяться у SOURCE_DIR.	На жаль, не вдалося виявити послідовність кроків, що призводять до появи цього дефекту.

Відтворюваність	Важливість	Терміновість	Симптом	Можливість обійти	Коментар
4	5	6	7	8	9
Іноді	Висока	Висока	Некоректна операція	Ні	

Хороший звіт, при написанні якого суть проблеми зрозуміла:

Короткий опис	Докладний опис	Кроки з відтворення
1	2	3
Програма не розрізняє файли та символічні посилання на файли.	Якщо в каталог SOURCE_DIR помістити символічне посилання на файл, виникає така помилкова поведінка: а) Якщо символічне посилання вказує на файл усередині SOURCE_DIR, файл обробляється двічі, а DESTINATION_DIR переміщається як сам файл, так і символічне посилання на нього. б) Якщо символічне посилання вказує на файл поза SOURCE_DIR, програма обробляє цей файл, переміщає символічне посилання і	1. У довільному місці створити таку структуру каталогів: /SRC/ /DST/ /X/ 2. Помістити в каталоги SRC та X кілька довільних файлів допустимого формату та розміру. 3. Створити в каталозі SRC два символічні посилання: а) на будь-який із файлів усередині каталогу SRC; б) на будь-який із файлів усередині каталогу X. 4. Запустити програму. Дефект: у каталог DST переміщені як файли, і символічні посилання; вміст каталогу X

	сам файл у DESTINATION_DIR, а потім продовжує обробляти файли в каталозі, в якому спочатку був оброблений файл. Act: додаток вважає символічні посилання файли самими файлами (див. подробиці вище). Exp: якщо у каталозі SOURCE_DIR програма виявляє символічне посилання, вона припиняє роботу з виведенням повідомлення «Symbolic link [ім'я символічного посилання] in SOURCE_DIR folder de-tected. Remove it manually or restart ap-plication with --force_file_operations key to remove automatically.»	оброблено та переміщено до каталогу DST.

Відтворюваність	Важливість	Терміновість	Симптом	Можливість обійти	Коментар
4	5	6	7	8	9
Завжди	Висока	Звичайна	Некоректна операція	Ні	Швидкий погляд код показав, що замість is_file() використовується file_exists(). Схоже, проблема у цьому. Також цей дефект призводить до спроби опрацювати каталоги як файли. У алгоритмі обробки

					SOURCE_DIR явно є логічна помилка: за жодних умов додаток має обробляти файли, що є поза SOURCE_DIR, тобто. щось не так з генерацією чи перевіркою повних імен файлів.

Відтворити дефект . Ця дія не тільки допоможе надалі правильно заповнити поле «Відтворюваність», а й дозволить уникнути неприємної ситуації, в якій за дефект програми буде прийнято якийсь короткочасний збій, який (швидше за все) стався десь у вашому комп'ютері або в іншій частині IT-інфраструктури, яка не має відношення до додатка, що тестується.

Перевірити наявність опису знайденого вами дефекту .

Обов'язково варто перевірити, чи немає в системі управління дефектами опису саме того дефекту, який ви щойно виявили. Це проста дія, що не стосується безпосередньо написання звіту про дефект, але значно скорочує кількість звітів, відхилених з резолюцією «дублікат».

Сформулювати суть проблеми . Формулювання проблеми у вигляді «що зробили (кроки з відтворення), що отримали (фактичний результат у докладному описі), що очікували отримати (очікуваний результат у докладному описі)» дозволяє не тільки підготувати дані для заповнення полів звіту, але ще краще зрозуміти суть проблеми.

У загальному випадку формула «що зробили, що отримали, що очікували отримати» хороша з наступних причин:

· Прозорість і зрозумілість: дотримуючись цієї формули, ви готуєте саме дані для звіту про дефект, не скочуючи в розлогі абстрактні міркування.

· Легкість верифікації дефекту: розробник, використовуючи ці дані, може швидко відтворити дефект, а тестувальник після виправлення дефекту переконатися, що справді виправлений.

· Очевидність для розробників: ще до спроби відтворення дефекту видно, чи насправді описане є дефектом, чи тестувальник десь помилився, записавши в дефекти коректну поведінку програми.

· Позбавлення зайвої безглуздої комунікації: докладні відповіді на «що зробили, що отримали, що очікували отримати» дозволяють вирішувати проблему та усувати дефект без необхідності запиту, пошуку та обговорення додаткових відомостей.

· Простота: на фінальних стадіях тестування із залученням кінцевих користувачів можна відчутно підвищити ефективність зворотного зв'язку, що надходить, якщо пояснити користувачам суть цієї формули і попросити їх дотримуватися її при написанні повідомлень про виявлені проблеми.

Інформація, отримана на даному етапі, стає фундаментом для всіх подальших дій щодо написання звіту.

Заповнити поля звіту . Поля звіту про дефект ми вже розглянули раніше, тепер лише підкреслимо, що починати найкраще з докладного опису, т.к. в процесі заповнення цього поля може виявитися безліч додаткових деталей, а також з'являться думки щодо формулювання стисненого та інформативного короткого опису.

Якщо ви знаєте, що для заповнення якогось поля у вас не вистачає даних, проведіть додаткове дослідження. Якщо й воно не допомогло, опишіть у відповідному полі (якщо воно текстове), чому ви утруднюєтеся з його заповненням, або (якщо поле є список) виберіть значення, яке, на вашу думку, найкраще характеризує проблему (у деяких випадках інструментальний засіб дозволяє вибрати значення на кшталт "невідомо", тоді виберіть його).

Якщо у вас немає ідей щодо усунення дефекту, або він настільки тривіальний, що не потребує подібних пояснень, не пишіть у коментарях «текст заради тексту»: коментарі виду «рекомендую усунути цей дефект» не просто безглузді, але ще й дратують.

Перечитати звіт (і ще раз перечитати звіт) . Після того, як все написано, заповнено та підготовлено, ще раз уважно перечитайте написане. Дуже часто ви зможете виявити, що в процесі доопрацювання тексту десь вийшли логічні нестиковки чи дублювання, десь вам захочеться покращити формулювання, десь щось змінити.

Ідеал недосяжний, і не варто витрачати вічність на один звіт про дефект, але й відправляти невичитаний документ - теж ознака поганого тону.

Після оформлення звіту про дефект рекомендується додатково дослідити ту область програми, в якій ви щойно виявили дефект. Практика показує, що дефекти часто проявляються групами.

4 . 7 . Оцінка трудовитрат, планування та звітність

4 . 7.1 . Планування та звітність . У розділі «Логіка створення ефективних перевірок» ми з прикладу «Конвертера файлів» міркували у тому, як із мінімальних трудовитратах отримати максимальний ефект від тестування. Це було просто, т.к. наш додаток смішно за своїми масштабами. Але давайте уявімо, що доводиться тестувати реальний проект, де вимоги в «сторінковому еквіваленті» займають сотні і навіть тисячі сторінок. Також давайте згадаємо розділ «Детальна класифікація тестування» з її кількома десятками видів тестування (і це без урахування того факту, що їх можна досить гнучко комбінувати, отримуючи нові варіанти) і подумаємо, як застосувати всі ці знання (і можливості, що відкриваються ними) у великому проект.

Навіть якщо припустити, що ми ідеально знаємо всі технічні аспекти майбутньої роботи, не відповідальними залишаються такі питання, як:

- Коли і з чого почати?
- Чи все необхідне для виконання роботи у нас є? Якщо ні, де взяти недостатнє?
- У якій послідовності виконувати різні види робіт?
- Як розподілити відповідальність між учасниками команди?
- Як організувати звітність перед зацікавленими особами?
- Як об'єктивно визначати прогрес та досягнуті успіхи?
- Як заздалегідь побачити можливі проблеми, щоб встигнути запобігти їм?
- Як організувати нашу роботу так, щоб за мінімуму витрат отримати максимум результату?

Ці та багато подібні до них питання вже лежать поза технічною галуззю — вони належать до управління проектом. Це завдання саме собою величезна, тому ми розглянемо лише малу її частина, з якою

багатьом тестувальникам доводиться мати справу, — планування і звітність.

Згадаймо життєвий цикл тестування: кожна ітерація починається з планування та закінчується звітністю, яка стає основою для планування наступної ітерації – і таке інше. Таким чином, планування та звітність перебувають у тісному взаємозв'язку, і проблеми з одним із цих видів діяльності неминуче призводять до проблем з іншим видом, а зрештою і до проблем із проектом загалом.

Якщо висловити цю думку чіткіше і за пунктами, виходить:

· Без якісного планування не ясно, кому і що потрібно робити.

· Коли не ясно, кому і що потрібно робити, робота виконується погано.

· Коли робота виконана погано та не зрозумілі точні причини, неможливо зробити правильні висновки про те, як виправити ситуацію.

· Без правильних висновків неможливо створити якісний звіт про результати роботи.

· Без якісного звіту про результати роботи неможливо створити якісний план подальшої роботи.

· Всі. Порочне коло замкнулося. Проект вмирає.

Здавалося б, так і в чому складність? Давайте добре плануватимемо і писатимемо якісні звіти — і все буде добре. Проблема в тому, що відповідні навички розвинені достатньо в украй невеликого відсотка людей.

Корінь проблеми полягає в тому, що плануванню та звітності в школах та університетах вчать досить поверхово, при цьому на практиці часто вихолощуючи ці поняття до порожньої формальності (планів, на які ніхто не дивиться, та звітів, які ніхто не читає; знову ж таки, кому -то пощастило побачити суворо обернену ситуацію, але явно небагатьом).

Розглянемо класичні визначення.

Планування - безперервний процес прийняття управлінських рішень та методичної організації зусиль щодо їх реалізації з метою забезпечення якості деякого процесу протягом тривалого часу.

До високорівневих завдань планування належать:

· Зниження невизначеності;

· підвищення ефективності;

· Поліпшення розуміння цілей;

· Створення основи для управління процесами.

Звітність — збирання та поширення інформації про результати роботи (включаючи поточний статус, оцінку прогресу та прогноз розвитку ситуації).

До високорівневих завдань звітності належать:

· Збір, агрегація та надання у зручній для сприйняття формі об'єктивної інформації про результати роботи;

· Формування оцінки поточного статусу та прогресу (у порівнянні з планом);

· Позначення існуючих та можливих проблем (якщо такі є);

· Формування прогнозу розвитку ситуації та фіксація рекомендацій щодо усунення проблем та підвищення ефективності роботи.

4 . 7.2 . Тест-план та звіт про результати тестування .

Тест-план — документ, що описує та регламентує перелік робіт із тестування, а також відповідні техніки та підходи, стратегію, галузі відповідальності, ресурси, розклад та ключові дати.

До низькорівневих завдань планування у тестуванні відносяться:

· Оцінка обсягу та складності робіт;

· Визначення необхідних ресурсів та джерел їх отримання;

· Визначення розкладу, термінів та ключових точок;

· Оцінка ризиків та підготовка превентивних контрзаходів;

· Розподіл обов'язків та відповідальності;

· Узгодження робіт із тестування з діяльністю учасників проектної команди, які займаються іншими завданнями.

Як і будь-який інший документ, тест-план може бути якісним або мати недоліки. Якісний тест-план має більшість властивостей якісних вимог, а також розширює їх набір наступними пунктами:

· Реалістичність (запланований підхід реально виконаємо).

· Гнучкість (якісний тест-план не тільки модифікується з точки зору роботи з документом, але і побудований таким чином, щоб при виникненні непередбачених обставин допускати швидку зміну будь-якої зі своїх частин без порушення взаємозв'язку з іншими частинами).

· Узгодженість із загальним проектним планом та іншими окремими планами (наприклад, планом розробки).

Тест-план створюється на початку проекту та доопрацьовується в міру потреби протягом усього часу життя проекту за участю найбільш кваліфікованих представників проектної команди, задіяних у забезпеченні якості. Відповідальним за створення тест-плану, як правило, є провідний тестувальник (тест-лід).

У випадку тест-план включає такі розділи.

Ціль . Гранично короткий опис мети розробки програми (частково це нагадує бізнес-вимоги, але інформація подається в ще більш стислому вигляді і в контексті того, на що слід звертати першорядну увагу при організації тестування і підвищення якості).

Області, що піддаються тестуванню . Перелік функцій або нефункціональних особливостей програми, які будуть тестовані. У деяких випадках тут наводиться пріоритет відповідної області.

Області, що не піддаються тестуванню . Перелік функцій або нефункціональних особливостей програми, які не будуть тестовані. Причини виключення тієї чи іншої області зі списку тестованих можуть бути різними — від гранично низької їх важливості для замовника до нестачі часу чи інших ресурсів. Цей перелік складається, щоб у проектної команди та інших зацікавлених осіб було чітке єдине розуміння, що тестування таких особливостей програми не заплановано . Такий підхід дозволяє виключити появу хибних очікувань та неприємних сюрпризів.

Тестова стратегія та підходи . Опис процесу тестування з погляду методів, підходів, видів тестування, технологій, інструментальних засобів і т.д.

Критерії . Цей розділ включає такі підрозділи:

- Приймальні критерії, критерії якості — будь-які об'єктивні показники якості, яким продукт, що розробляється, повинен відповідати з точки зору замовника або користувача, щоб вважатися готовим до експлуатації.
- Критерії початку тестування — перелік умов, під час виконання яких команда розпочинає тестування. Наявність цього критерію страхує команду від безглуздої витрати зусиль за умов, коли тестування не принесе очікуваної користі.
- Критерії зупинення тестування — перелік умов, під час яких тестування припиняється. Наявність цього критерію

також страхує команду від безглуздої витрати зусиль за умов, коли тестування не принесе очікуваної користі.

- Критерії відновлення тестування — перелік умов, під час яких тестування відновлюється (зазвичай, після призупинення).
- Критерії завершення тестування – перелік умов, при виконанні яких тестування завершується. Наявність цього критерію страхує команду як від передчасного припинення тестування, і від продовження тестування за умов, коли вже перестає приносити відчутний ефект.

Ресурси . У цьому розділі тест-плану перераховуються всі необхідні для успішної реалізації стратегії тестування ресурси, які можна розділити на:

- програмні ресурси (яке ПЗ необхідне команді тестувальників, скільки копій та з якими ліцензіями (якщо йдеться про комерційне ПЗ));
- апаратні ресурси (яке апаратне забезпечення, в якій кількості та до якого моменту необхідно команді тестувальників);
- людські ресурси (скільки фахівців якого рівня та зі знаннями в яких галузях має підключитися до команди тестувальників у той чи інший момент часу);
- тимчасові ресурси (скільки за часом займе виконання тих чи інших робіт);
- фінансові ресурси (у яку суму обійдеться використання наявних або отримання ресурсів, що перебувають у попередніх пунктах цього списку); у багатьох компаніях фінансові ресурси можуть бути окремим документом, т.к. є конфіденційною інформацією.

Розклад . Фактично це календар, в якому зазначено, що і на який момент має бути зроблено. Особлива увага приділяється т.зв. «ключовим точкам», на момент настання яких має бути отриманий якийсь значний відчутний результат.

Ролі та відповідальність . Перелік необхідних ролей (наприклад, «провідний тестувальник», «експерт з оптимізації продуктивності») та сфера відповідальності фахівців, які виконують ці ролі.

Оцінка ризиків . Перелік ризиків, які з ймовірністю можуть виникнути в процесі роботи над проектом. По кожному ризику дається оцінка загрози, що їм представляється, і наводяться варіанти виходу з ситуації.

Документація . Перелік тестової документації, що використовується, із зазначенням, хто і коли повинен її готувати і кому передавати.

Метрики . Числові характеристики показників якості, способи їхньої оцінки, формули і т.д. На цей розділ зазвичай формується безліч посилань з інших розділів тест-плану.

Метрики у тестуванні є настільки важливими, що про них ми поговоримо окремо.

Метрика – числова характеристика показника якості. Може включати опис способів оцінки та аналізу результату.

Спочатку пояснимо важливість метрик на тривіальному прикладі. Уявіть, що замовник цікавиться поточним станом справ та просить вас коротко охарактеризувати ситуацію із тестуванням на проекті. Загальні слова в стилі «все добре», «все погано», «нормально» тощо, звичайно, не влаштують, т.к. вони гранично суб'єктивні і може бути вкрай далекі від реальності. І зовсім інакше виглядає відповідь на кшталт такої: «Реалізовано 79% вимог (у т.ч. 94% важливих), за останні три спринти тестове покриття виросло з 63% до 71%, а загальний показник проходження тест-кейсів зріс із 85% до 89%. Іншими словами, ми повністю вкладаємося в план за всіма ключовими показниками, а з розробки навіть йдемо з невеликим випередженням».

Щоб оперувати всіма цими числами (а вони потрібні не тільки для звітності, а й для роботи проектної команди), їх потрібно якось обчислити. Саме це дозволяють зробити метрики. Потім обчислені значення можна використовувати для:

- прийняття рішень про початок, призупинення, відновлення або припинення тестування;
- Визначення ступеня відповідності продукту заявленим критеріям якості;
- Визначення ступеня відхилення фактичного розвитку проекту від плану;
- Виявлення «вузьких місць», потенційних ризиків та інших проблем;

- Оцінки результативності прийнятих управлінських рішень;
- Підготовки об'єктивної інформативної звітності;
- і т.д.

Метрики може бути як прямими (не вимагають обчислень), і розрахунковими (обчислюються за формулою). Типові приклади прямих метрик – кількість розроблених тест-кейсів, кількість знайдених дефектів тощо. У розрахункових метриках можуть використовуватися як цілком тривіальні, і досить складні формули (у цьому курсі ми їх розглядати не будемо).

У тестуванні існує велика кількість загальноприйнятих метрик, багато з яких можна зібрати автоматично з використанням інструментальних засобів управління проектами. Наприклад:

- Відсоткове відношення (не) виконаних тест-кейсів до всіх наявних;
- Відсотковий показник успішного проходження тест-кейсів;
- Відсотковий показник заблокованих тест-кейсів;
- Щільність розподілу дефектів;
- Ефективність усунення дефектів;
- Розподіл дефектів по важливості та терміновості;
- і т.д.

Як правило, при формуванні звітності нас буде цікавити не тільки поточне значення метрики, але і її динаміка в часі, яку дуже зручно зображати графічно (що теж можуть автоматично виконувати багато засобів управління проектами).

Таким чином, ми бачимо, що метрики є найпотужнішим засобом збирання та аналізу інформації. І разом з тим тут криється небезпека: за жодних умов не можна допускати ситуації «метрик заради метрик», коли інструментальний засіб збирає безліч даних, обчислює безліч чисел і будує десятки графіків, але... ніхто не розуміє, як їх трактувати.

І, нарешті, варто згадати про звані «метрики покриття», т.к. вони дуже часто згадуються у різній літературі.

Покриття - процентний вираз ступеня, в якому досліджуваний елемент торкнуться відповідним набором тест-кейсів.

Найпростішими представниками метрик покриття можна вважати:

- Метрику покриття вимог (вимога вважається «покритим», якщо на нього посилається хоча б один тест-кейс) .

· Метрику щільності покриття вимог (враховується, скільки тест-кейсів посилається на кілька вимог) .

· Метрику покриття класів еквівалентності (аналізується, скільки класів еквівалентності порушено тест-кейсами) .

· Метрику покриття граничних умов (аналізується, скільки значень із групи граничних умов порушено тест-кейсами):

· Метрики покриття коду модульними тест-кейсами. Таких метрик дуже багато, але вся їх суть зводиться до виявлення певної характеристики коду (кількість рядків, гілок, шляхів, умов і т.д.) та визначення, який відсоток представників цієї характеристики покритий тест-кейсами.

Перейдемо наприклад - навчальному тест-плану для нашого додатка «Конвертер файлів» . Нагадаємо, що додаток є гранично простим, тому і тест-план буде дуже маленьким (проте, зверніть увагу, наскільки значну його частину займатиме розділ метрик).

4.7.3. Приклад тест-плану

Для того, щоб заповнити деякі частини тест-плану, нам доведеться зробити припущення про склад проектної команди та час, відведений на розробку проекту.

Ціль . Коректне автоматичне перетворення вмісту документів до єдиного кодування з продуктивністю, що значно перевищує продуктивність людини при виконанні аналогічного завдання.

Області, що піддаються тестуванню .

· ПТ-1. *: димовий тест.

· ПТ-2. *: димовий тест, тест критичного шляху.

· ПТ-3. *: тест критичного шляху.

· БП-1. *: димовий тест, тест критичного шляху.

· АК-2. *: димовий тест, тест критичного шляху.

· О-4: димовий тест.

· Про-5: димовий тест.

· ДС- *: димовий тест, тест критичного шляху.

Області, що не піддаються тестуванню

· СХ-1: програма розробляється як консольна.

· СХ-2, О-1, О-2: програма розробляється на PHP зазначеної версії.

· АК-1.1: заявлена характеристика знаходиться поблизу нижньої межі продуктивності операцій, характерних для програми, що розробляється.

О-3: не потребує реалізації.

О-6: не потребує реалізації.

Тестова стратегія та підходи

Загальний підхід.

Специфіка роботи програми полягає в одноразовому конфігуруванні досвідченим фахівцем та подальшому використанні кінцевими користувачами, для яких доступна лише одна операція - розміщення файлу в каталозі-приймачі. Тому питання зручності використання, безпеки тощо. не досліджуються у процесі тестування.

Рівні функціонального тестування:

· Димовий тест: автоматизований з використанням командних файлів ОС Windows та Linux.

· Тест критичного шляху: виконується вручну.

· Розширений тест немає, т.к. для даної програми ймовірність виявлення дефектів на цьому рівні дуже мала.

В силу кросфункціональності команди значного вкладу у підвищення якості очікується від аудиту коду, суміщеного з ручним тестуванням за методом білої скриньки. Автоматизація тестування коду не буде застосовуватись через вкрай обмежений час.

Критерії

· Приймальні критерії: успішне проходження 100% тест-кейсів рівня димового тестування та 90% тест-кейсів рівня критичного шляху за умови усунення 100% дефектів критичної та високої важливості. Підсумкове покриття вимог тест-кейсами має становити щонайменше 80 %.

· Критерії початку тестування: вихід білда.

· Критерії припинення тестування: перехід до тесту критичного шляху допустимо лише при успішному проходженні 100% тест-кейсів димового тесту; тестування може бути призупинено у разі, якщо при

виконанні не менше 25 % запланованих тест-кейсів більше 50 % завершилися виявленням дефекту.

· Критерії відновлення тестування: виправлення понад 50% виявлених на попередній ітерації дефектів.

· Критерії завершення тестування: виконання понад 80% запланованих на ітерацію тест-кейсів.

Ресурси

· Програмні ресурси: чотири віртуальні машини (дві з ОС Windows 7 Ent x64, дві з ОС Linux Ubuntu 14 LTS x64), дві копії PHP Storm 8.

· Апаратні ресурси: дві стандартні робочі станції (8GB RAM, i7 3GHz).

Людські ресурси:

Старший розробник з досвідом тестування (100% зайнятість на всьому протязі проекту). Ролі на проекті: лідер команди, старший розробник.

Тестувальник зі знанням PHP (100% зайнятість на всьому протязі проекту). Роль на проекті: тестувальник.

Тимчасові ресурси : один робочий тиждень (40 годин).

Фінансові ресурси : згідно із затвердженим бюджетом. Додаткові фінансові ресурси не потрібні.

Розклад

· 25.05 - формування вимог.

· 26.05 – розробка тест-кейсів та скриптів для автоматизованого тестування.

· 27.05-28.05 - основна фаза тестування (виконання тест-кейсів, написання звітів про дефекти).

· 29.05 - завершення тестування та підбиття підсумків.

Ролі та відповідальність

· Старший розробник: участь у формуванні вимог, участь у аудиті коду.

· Тестувальник: формування тестової документації, реалізація тестування, участь в аудиті коду.

Оцінка ризиків

· Персонал (імовірність низька): у разі непрацездатності будь-якого з учасників команди можна звернутися до представників проекту «Каталогізатор» для надання тимчасової заміни (договореність з менеджером «Каталогизатора» Джоном Смітом досягнуто).

· Час (імовірність висока): замовником позначено крайній термін здачі 01.06, тому час є критичним ресурсом. Рекомендується докласти максимум зусиль до того, щоб фактично завершити проект 28.05 для того, щоб один день (29.05) залишився в запасі.

· Інші ризики: інших специфічних ризиків не виявлено.

Документація

· Вимоги. Відповідальний - тестувальник, дата готовності 25.05.

· Тест-кейси та звіти про дефекти. Відповідальний - тестувальник, період створення 26.05-28.05.

· Звіт про результати тестування. Відповідальний - тестувальник, дата готовності 29.05.

Метрики . Тут наводяться розрахункові формули.

На цьому ми завершуємо обговорення планування та переходимо до звітності, яка завершує цикл тестування.

4.7.4. Звіт про результати тестування

Звіт про результати тестування — документ, що узагальнює результати робіт із тестування та містить інформацію, достатню для співвіднесення поточної ситуації з тест-планом та прийняття необхідних управлінських рішень.

До низькорівневих завдань звітності у тестуванні належать:

· Оцінка обсягу та якості виконаних робіт;

· Порівняння поточного прогресу з тест-планом (у тому числі за допомогою аналізу значень метрик);

· Опис наявних складнощів та формування рекомендацій щодо їх усунення;

· Надання особам, зацікавленим у проекті, повної та об'єктивної інформації про поточний стан якості проекту, вираженої в конкретних фактах та числах.

Як і будь-який інший документ, звіт про результати тестування може бути якісним або мати недоліки. Якісний звіт про результати тестування має багато властивостей якісних вимог, а також розширює їх набір наступними пунктами:

· Інформативність (в ідеалі після прочитання звіту не повинно залишатися жодних відкритих питань про те, що відбувається з проектом у контексті якості).

· Точність та об'єктивність (за жодних умов у звіті не допускається спотворення фактів, а особисті думки повинні бути підкріплені твердими обґрунтуваннями).

Звіт про результати тестування створюється за заздалегідь обумовленим розкладом (залежним від моделі управління проектом) за участю більшості представників проектної команди, задіяних у забезпеченні якості. Велика кількість фактичних даних для звіту може бути легко витягнута у зручній формі із системи управління проектом. Відповідальним за створення звіту, як правило, є провідний тестувальник (тест-лід). За необхідності звіт може обговорюватися на невеликих зборах.

Звіт про результати тестування насамперед потрібний наступним особам:

· Менеджеру проекту - як джерело інформації про поточну ситуацію та основа для прийняття управлінських рішень;

· Керівнику команди розробників («дев-ліду») - як додатковий об'єктивний погляд на те, що відбувається на проекті;

· Керівнику команди тестувальників («тест-лід») - як спосіб структурувати власні думки і зібрати необхідний матеріал для звернення до менеджера проекту з насущних питань, якщо в цьому є необхідність;

· Замовнику - як найбільш об'єктивне джерело інформації про те, що відбувається на проекті, за який він платить свої гроші.

У випадку звіт про результати тестування включає такі розділи .

· Короткий опис. У гранично короткій формі відображає основні досягнення, проблеми, висновки та рекомендації. В ідеальному випадку прочитання короткого опису може бути достатньо для

формування повноцінного уявлення про те, що позбавить необхідності читати весь звіт (це важливо, тому що звіт про результати тестування може потрапляти в руки дуже зайнятим людям).

· Команда тестувальників. Список учасників проектної команди, задіяних у забезпеченні якості, із зазначенням їх посад та ролей у підзвітний період.

· Опис процесу тестування. Послідовний опис того, які роботи було виконано за підзвітний період.

· Розклад. Детальний розклад роботи команди тестувальників та/або особисті розклади учасників команди.

· Статистика щодо нових дефектів. Таблиця, в якій представлені дані щодо виявлених за підзвітний період дефектів (з класифікацією по стадії життєвого циклу та важливості).

· Список нових дефектів. Список виявлених за підзвітний період дефектів зі своїми короткими описами та важливістю.

· Статистика за всіма дефектами. Таблиця, в якій представлені дані щодо виявлених за весь час існування проекту дефектів (з класифікацією по стадії життєвого циклу та важливості). Як правило, в цей же розділ додається графік, що відображає такі статистичні дані.

· Рекомендації. Обґрунтовані висновки та рекомендації щодо прийняття тих чи інших управлінських рішень (зміни тест-плану, запиту чи звільнення ресурсів тощо). Тут цієї інформації можна відвести більше місця, ніж у короткому описі (summary), наголосивши саме на тому, що і чому рекомендується зробити в наявній ситуації.

· Додатки. Фактичні дані (як правило, значення метрик та графічне уявлення їх зміни у часі).

4.7.5. Логіка побудови звіту про результати тестування

Для того, щоб звіт про результати тестування був дійсно корисним, при його створенні слід постійно пам'ятати про універсальну логіку звітності (рис. 1) . 4.6), особливо актуальною для таких розділів звіту про результати тестування, як короткий опис і рекомендації:

· Висновки будуються на основі цілей (які були відображені у плані).

· Висновки доповнюються рекомендаціями.

· Як висновки, і рекомендації суворо обгрунтовуються.

· Обґрунтування спирається на об'єктивні факти.

Рисунок 4.6 - Універсальна логіка звітності

Висновки мають бути:

Короткими . Порівняйте:

Погано	добре
1.17. Як показав глибокий аналіз протоколів про виконання тестування, можна зробити досить впевнені висновки про те, що переважна більшість функцій, зазначених замовником як найбільш важливі, функціонує в рамках допустимих відхилень від узгоджених на останньому обговоренні із замовником метрик якості.	1.11. Базова функціональність є повністю працездатною (див. 2.1–2.2). 1.23. Існують некритичні проблеми з деталізацією повідомлень у файлі журналу (див. 2.3–2.4). 1.28. Тестування програми під ОС Linux не вдалося провести через відсутність сервера SR-85 (див. 2.5).

Інформативними . Порівняйте:

Погано	добре
1.8. Результати обробки файлів з множинними кодуваннями, представленими в порівнянні пропорціях, залишають бажати кращого. 1.9. Додаток не запускається за деяких значеннях параметрів командного рядка. 1.10. Незрозуміло, що відбувається з аналізом зміни вмісту вхідного каталогу.	1.8. Виявлено серйозні проблеми з бібліотекою розпізнавання кодувань (див. BR 834). 1.9. Порушено функціональність аналізу параметрів командного рядка (див. BR 745, BR 877, BR 878). 1.10. Виявлено нестабільність у роботі модуля «Сканер», проводяться додаткові дослідження.

Корисними для читача звіт . Порівняйте:

Погано	добре
1.18. Деякі тести пройшли напрочуд добре.	Представленого в колонці «Погано» просто не повинно бути у звіті!

1.19. У процесі тестування ми не мали складності з налаштуванням середовища автоматизації. 1.20. Порівняно з результатами, які були отримані вчора, ситуація трохи покращала. 1.21. З якістю, як і раніше, є деякі проблеми. 1.22. Частина команди була у відпустці, але ми все одно впоралися.	

Рекомендації мають бути:

Короткими . Так, ми знову говоримо про стислості, т.к. її відсутністю страждає дуже багато документів. Порівняйте:

Погано	добре
2.98. Ми рекомендуємо розглянути можливі варіанти виправлення цієї ситуації в контексті пошуку оптимального рішення за умови мінімізації зусиль розробників та максимального підвищення відповідності додатку заявленим критеріям якості, а саме: дослідити можливість заміни деяких бібліотек на їх якісніші аналоги.	2.98. Необхідно змінити спосіб визначення кодування тексту у документі. Можливі рішення: [складно, надійно, але дуже довго] написати власне рішення; [вимагає додаткового дослідження та узгодження] замінити проблемну бібліотеку «cflk_n_r_coding» аналогом (можливо, комерційним).

Реально здійсненними. Порівняйте:

Погано	добре
2.107. Використовувати механізм обробки слів, аналогічний використовуваному у Google. 2.304. Не завантажуйте в оперативну пам'ять інформацію про файли у вхідному каталозі. 2.402. Повністю переписати проект без використання зовнішніх бібліотек.	2.107. Реалізувати алгоритм приведення слів української мови до називного відмінка (див. опис за посиланням …) 2.304. Збільшити обсяг доступної скрипту оперативної пам'яті на 40-50% (в ідеалі - до 512 МБ). 2.402. Замінити власними рішеннями функції аналізу вмісту каталогу та параметрів файлів бібліотеки "cflk_n_r_flstm".

Ті, хто дає як розуміння того, що треба зробити, так і деякий простір для прийняття власних рішень. Порівняйте:

Погано	добре
2.212. Рекомендуємо пошукати варіанти вирішення цього питання. 2.245. Використовувати лише дискове сортування. 2.278. Виключити можливість надсилання некоректних імен файлу журналу через параметр командного рядка.	2.212. Можливі варіанти розв'язання: а) … б) [рекомендуємо!] … в) … 2.245. Додати функціональність визначення оптимального методу сортування залежно кількості доступної оперативної пам'яті. 2.278. Додати фільтрацію імені файлу журналу, який отримується через параметр командного рядка, за допомогою регулярного виразу.

Обґрунтування висновків та рекомендацій — проміжна ланка між гранично стислими результатами аналізу та величезною кількістю фактичних даних. Воно дає відповіді на запитання на кшталт:

- «Чому ми так вважаємо?»
- «Невже це так?!»
- «Де взяти додаткові дані?»

Порівняйте:

Погано	добре
4.107. Покриття вимог тест-кейс досить. 4.304. Потрібно більше зусиль направити на регресійне тестування. 4.402. Від скорочення термінів розробки варто відмовитись.	4.107. Покриття вимог тест-кейсами вийшло на достатній рівень (значення RC склало 63% при заявленому мінімумі 60% для поточної стадії проекту). 4.304. Потрібно більше зусиль направити на регресійне тестування, т.к. дві попередні ітерації виявили 21 дефект високої важливості (див. список 5.43) у функціональності, в якій раніше не виявлялося проблем. 4.402. Від скорочення термінів розробки слід відмовитися, т.к. поточне випередження графіка на 30 людино-годин може бути легко поглинено на стадії реалізації вимог R84* і R89*.

Фактичний матеріал містить найрізноманітніші дані, отримані у процесі тестування. Сюди можуть відноситися звіти про дефекти, журнали засобів автоматизації, створені різними додатками набори файлів і т.д. Як правило, до звіту про результати тестування додаються лише скорочені агреговані вибірки подібних даних (якщо це можливо), а також наводяться посилання на відповідні документи, розділи системи управління проектом, шляхи у сховище даних і т.д.