Функціональне тестування ПЗ та основи тест-дизайну

Лекція 1

Що таке тестування та як воно з’явилось

Тестування програмного забезпечення - процес аналізу програмного засобу та супутньої документації з метою виявлення дефектів і підвищення якості продукту.

У глосарії ISTQB немає терміна «тестування ПЗ», який широко використовується в українській мові. Там є лише термін «тестування (testing) ».

Що таке тестування та як воно з’явилось

50-60 debugging + exhaustive testing

70 positive testing + negative testing

80 software lifecycle

90 quality assurance

00 test-driven development

Хто такий тестувальник і що він робить

	Невеликі фірми	**Великі фірми**
Низька кваліфікація	Підмайстер, часно залишений сам-на-сам у вирішенні завдань.	Рядовий учасник проектів, одночасно проходить інтенсивне підвищення кваліфікації.
Висока кваліфікація	Майстер на всі руки з багатим, але не завжди структурованим досвідом	Експерт в одній або декількох областях, консультант, керівник напрямку.

Технічні навики

0. Знання іноземної мови. English – must have!
1. Впевнений користувач ПК.
2. Програмування. (JS)
3. Бази даних
4. Розуміння принципів роботи мереж та операційних систем
5. Розуміння принципів роботи веб-застосувань та мобільних застосувань

Особистісні якості

1) підвищена відповідальність і старанність;

2) гарні комунікативні навички, здатність ясно, швидко, чітко висловлювати свої думки;

3) терпіння, посидючість, уважність до деталей, спостережливість;

4) хороше абстрактне і аналітичне мислення;

5) здатність ставити нестандартні експерименти, схильність до дослідницької діяльності.

Навички

- Професійні - це ключові навички, що відрізняють тестувальника від інших IT-фахівців.
- Технічні - це загальні навички в області IT, якими проте повинен володіти і тестувальник.
- Особистісні - «soft skills».

Професійні навички

Процеси тестування і розробки програмного забезпечення

Процес тестування ПЗ - глибоке розуміння стадій процесу тестування, їх взаємозв'язку і взаємовпливу, вміння планувати власну роботу в рамках отриманого завдання в залежності від стадії тестування

Процес розробки ПЗ - загальне розуміння моделей розробки ПО, їхзв'язку з тестуванням, вміння розставляти пріоритети в своїй роботі в залежності від стадії розвитку проекту

Професійні навички

Робота з документацією

Аналіз вимог - вміння визначати взаємозв'язки і взаємозалежність між різними рівнями і формами представлення вимог, вміння формулювати питання з метою уточнення незрозумілих моментів

Тестування вимог - знання властивостей хороших вимог і наборів вимог, вміння аналізувати вимоги з метою виявлення їх недоліків, вміння усувати недоліки у вимогах, вміння застосовувати техніки підвищення якості вимог

Керування вимогами - загальне розуміння процесів виявлення, документування, аналізу та модифікації вимог

Професійні навички

Робота з документацією

Бізнес-аналіз - загальне розуміння процесів виявлення та документування різних рівнів і форм представлення вимог

Оцінка і планування

Створення тест-плану - загальне розуміння принципів планування в контексті тестування, вміння використовувати готовий тест-план для планування власної роботи

Створення стратегії тестування - загальне розуміння принципів побудови стратегії тестування, вміння використовувати готову стратегію для планування власної роботи

Оцінка трудозатрат - загальне розуміння принципів оцінки трудозатрат, вміння оцінювати власні трудозатрати при плануванні власної роботи

Професійні навички

Робота з тест-кейсами

Створення чек-листів - вміння використовувати техніку і підходи до проектування тестових випробувань, вміння декомпозувати тестовані об'єкти і поставлені завдання, вміння створювати чек-листи

Створення тест-кейсів - вміння оформляти тест-кейси згідно прийнятим шаблонами, вміння аналізувати готові тест-кейси, виявляти і усувати наявні в них недоліки

Управління тест-кейсами - загальне розуміння процесів створення, модифікації та підвищення якості тест-кейсів

Професійні навички

Методологія тестування

Функціональне і доменне тестування - знання видів тестування, тверде вміння використовувати техніку і підходи до проектування тестових випробувань, вміння створювати чек-листи і тест-кейси, вміння створювати звіти про дефекти

Тестування інтерфейса користувача - вміння проводити тестування інтерфейсу користувача на основі готових тестових сценаріїв або в рамках дослідницького тестування

Дослідницьке тестування - загальне вміння використовувати матриці для швидкого визначення сценаріїв тестування, загальне вміння проводити нові тести на основі результатів тільки що виконаних

Професійні навички

Методологія тестування

Інтеграційне тестування - вміння проводити інтеграційне тестування на основі готових тестових сценаріїв

Локалізаційне тестування - вміння проводити локалізаційного тестування на основі готових тестових сценаріїв

Інсталяційне тестування - вміння проводити інсталяційне тестування на основі готових тестових сценаріїв

Регресійне тестування - загальне розуміння принципів організації регресійного тестування, вміння проводити по готовим планам


Тестування
1. Загальне планування та аналіз вимог
2. Уточнення критеріїв прийому
3. Уточнення стратегії тестування
4. Розробка тест-кейсів
5. Виконання тест-кейсів
6. Фіксація знайдених дефектів
7. Аналіз результатів тестування
8. Звітність

Стадія 1

(Загальне планування і аналіз вимог) об'єктивно необхідна як мінімум для того, щоб мати відповідь на такі питання, як: що нам належить тестувати; як багато буде роботи; які є складності; чи все необхідне у нас є і т.д. Як правило, отримати відповіді на ці питання неможливо без аналізу вимог, тому що саме вимоги є первинним джерелом відповідей.

Стадія 2

(Уточнення критеріїв приймання) дозволяє сформулювати або уточнити метрики і ознаки можливості або необхідності початку тестування (entry criteria), припинення (suspension criteria) і відновлення (resumption criteria) тестування, завершення або припинення тестування (exit criteria).

Стадія 3

(Уточнення стратегії тестування) являє собою ще одне звернення до планування, але вже на локальному рівні: розглядаються і уточнюються ті частини стратегії тестування (test strategy), які є актуальними для поточної ітерації.

Стадія 4

(Розробка тест-кейсів) присвячена розробці, перегляду, уточненню, доопрацюванню, переробці та іншим діям з тест-кейсами, наборами тест-кейсів, тестовими сценаріями та іншими артефактами, які будуть використовуватися при безпосередньому виконанні тестування.

Стадія 5 та 6

(Виконання тест-кейсів) і стадія 6 (фіксація знайдених дефектів) тісно пов'язані між собою і фактично виконуються паралельно: дефекти фіксуються відразу по факту їх виявлення в процесі виконання тест-кейсів. Однак найчастіше після виконання всіх тест-кейсів і написання всіх звітів про знайдених дефектах проводиться явно виділена стадія уточнення, на якій всі звіти про дефекти розглядаються повторно з метою формування єдиного розуміння проблеми та уточнення таких характеристик дефекту, як важливість і терміновість.

Стадія 7 та 8

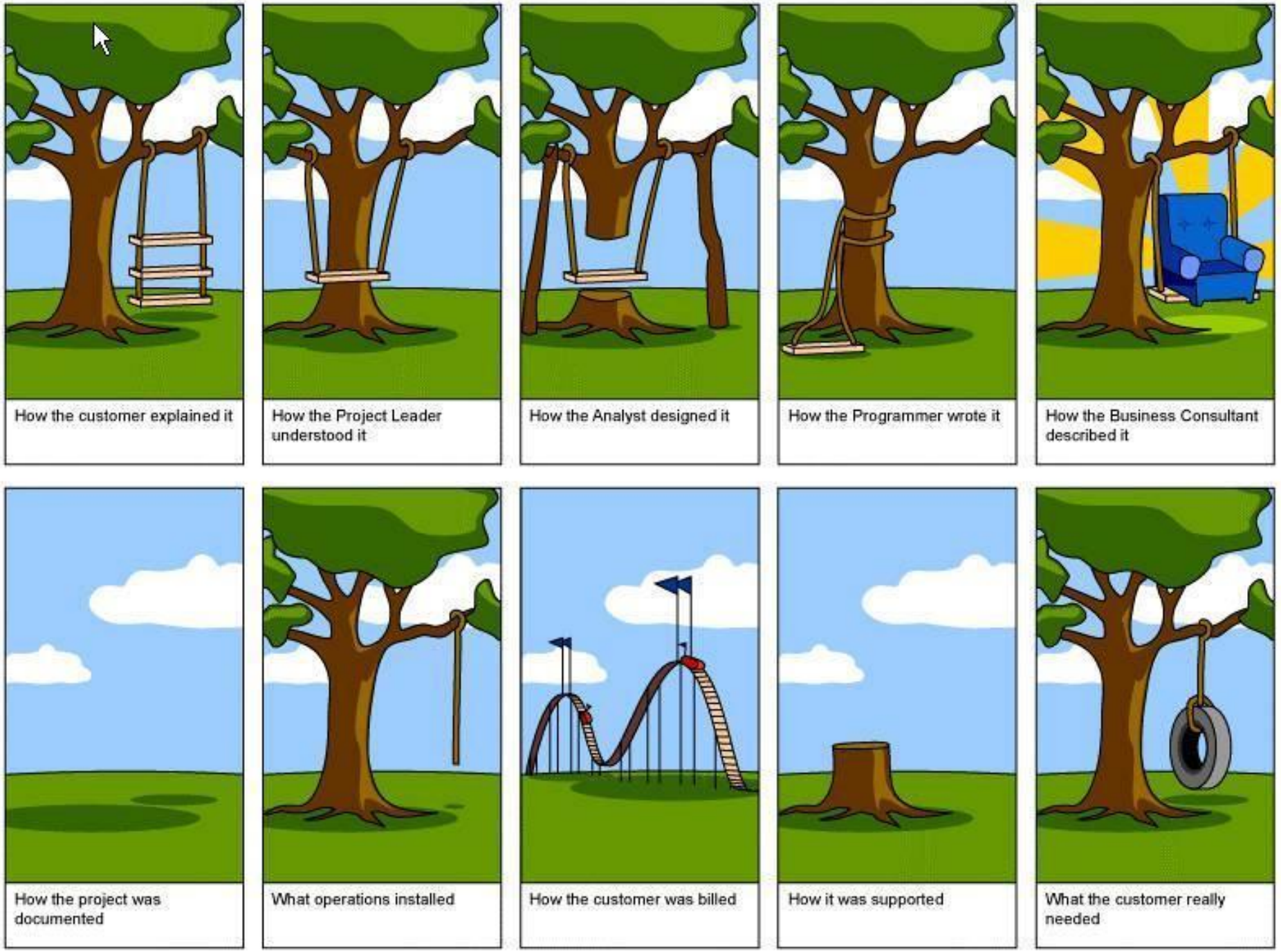
(Аналіз результатів тестування) і стадія 8 (звітність) також тісно пов'язані між собою і виконуються практично паралельно. Сформульовані на стадії аналізу результатів висновки безпосередньо залежать від плану тестування, критеріїв приймання та уточненої стратегії, отриманих на стадіях 1, 2 і 3. Отримані висновки оформляються на стадії 8 і служать основою для стадій 1, 2 і 3 наступній ітерації тестування. Таким чином, цикл замикається.

Вимоги

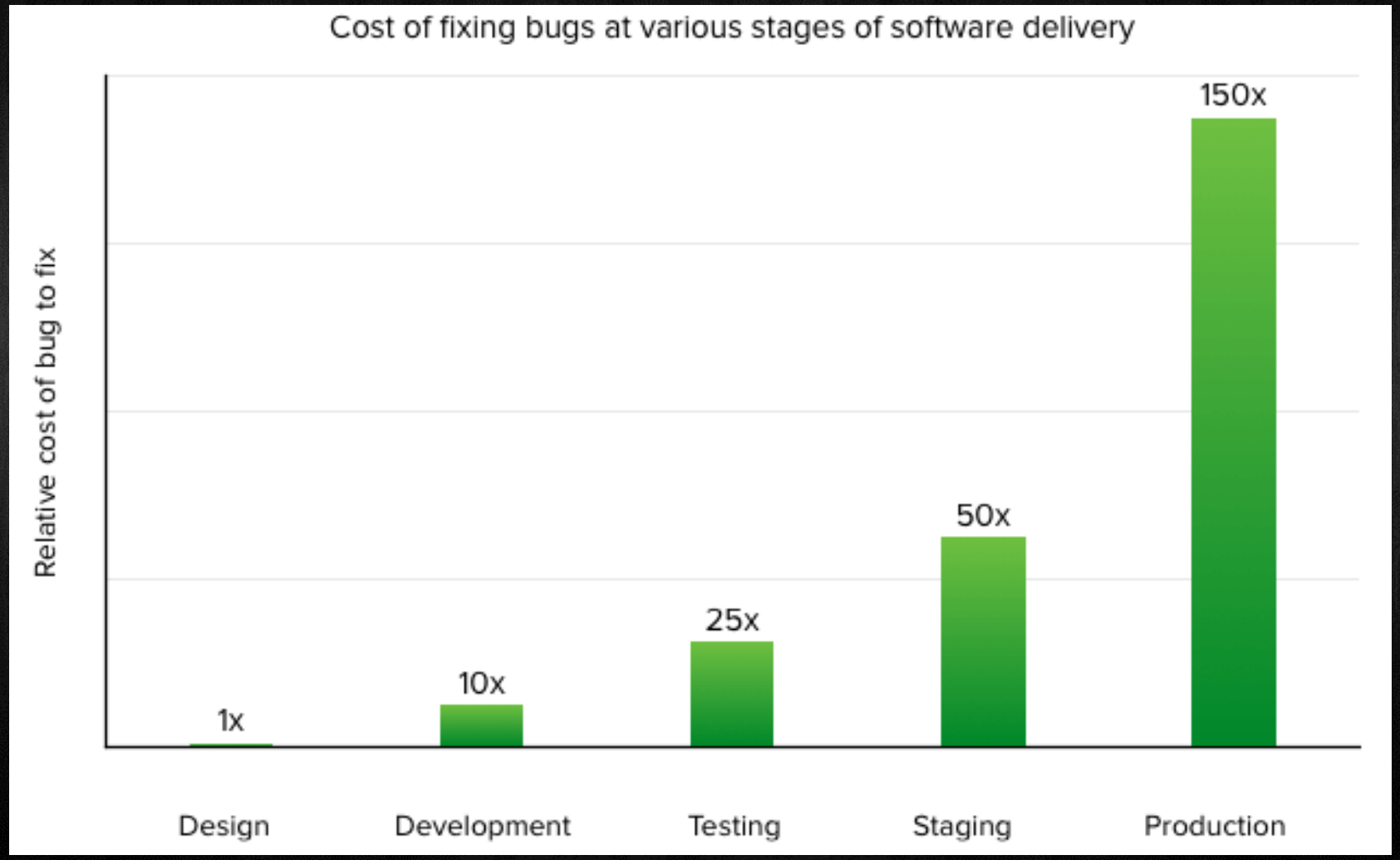
Вимога (requirement) - опис того, які функції і з дотриманням яких умов має виконувати додаток в процесі вирішення корисної для користувача функції.

Важливість вимог

- Дозволяють зрозуміти, що і з дотриманням яких умов система повинна робити.
- Надають можливість оцінити масштаб змін і управляти змінами.
- Єосновою для формування плану проекту (в тому числі плану тестування).
- Допомагають запобігати або вирішувати конфліктні ситуації.
- Спрощують розстановку пріоритетів в наборі завдань.
- Дозволяють об'єктивно оцінити ступінь прогресу в розробці проекту.


How the customer explained it
How the Project Leader understood it
How the Analyst designed it
How the Programmer wrote it
How the Business Consultant described it
How the project was documented
What operations installed
How the customer was billed
How it was supported
What the customer really needed

Вартість виправлення помилки



Джерела та шляхи виявлення вимог

Інтерв’ю	Робота з фокусними групами	Анкетування
Семінари та мозковий штурм	Спостереження	Прототипування
Аналіз документів	Моделювання процесів та взаємодій	Самостійний опис

Інтерв’ю

Найбільш універсальний шлях виявлення вимог, що полягає в спілкуванні проектного фахівця (як правило, фахівця з бізнес аналізу) і представника замовника (або експерта, користувача і т.д.) Інтерв'ю може проходити в класичному розумінні цього слова (бесіда у вигляді «питання-відповідь»), у вигляді переписки і т.д. Головним тут є те, що ключовими фігурами виступають двоє – той в кого беруть інтерв'ю і інтерв'юер (хоча це і не виключає наявності «аудиторії слухачів», наприклад, у вигляді осіб, поставлених в копію переписки).

Робота з фокусними групами

Може виступати як варіант «розширеного інтерв'ю», де джерелом інформації є не одна особа, а група осіб (Як правило, представляють собою цільову аудиторію, і / або які мають важливою для проекту інформацією, і / або уповноважених приймати важливі для проекту рішення).

Анкетування

Цей варіант виявлення вимог викликає багато суперечок, тому що за невірної реалізації може привести до нульового результату при об'ємних витратах. У той же час при правильній організації анкетування дозволяє автоматично зібрати і обробити величезну кількість відповідей від величезної кількості респондентів. Ключовим фактором успіху є правильне складання анкети, правильний вибір аудиторії і правильне представлення анкети.

Семінари та мозковий штурм

Семінари дозволяють групі людей дуже швидко обмінятися інформацією (і наочно продемонструвати ті чи інші ідеї), а також добре поєднуються з інтерв'ю, анкетуванням, прототипуванням і моделюванням - в тому числі для обговорення результатів та формування висновків і рішень. Мозковий штурм може проводитися і як частина семінару, і як окремий вид діяльності. Він дозволяє за мінімальний час згенерувати велику кількість ідей, які в подальшому можна не поспішаючи розглянути з точки зору їх використання для розвитку проекту.

Спостереження

Може виражатися як в буквальному спостереженні за деякими процесами, так і у включенні проектного фахівця в ці процеси як учасника. З одного боку, спостереження дозволяє побачити те, про що (по абсолютно різних міркувань) можуть промовчати ті, в кого беруть інтерв'ю, анкетовані і представники фокус-груп, але з іншого - забирає дуже багато часу і найчастіше дозволяє побачити лише частину процесів.

Прототипування

Полягає в демонстрації і обговоренні проміжних версій продукту (наприклад, дизайн сторінок сайту може бути спочатку представлений у вигляді картинок, і лише потім зверстаний). Це один з кращих шляхів пошуку єдиного розуміння і уточнення вимог, однак він може привести до серйозних додаткових витрат при відсутності спеціальних інструментів (що дозволяють швидко створювати прототипи) і занадто ранньому застосуванні (коли вимоги ще не стабільні, і висока ймовірність створення прототипу, що має мало спільного з тим, що хотів замовник).

Аналіз документів

Добре працює тоді, коли експерти з предметної області (тимчасово) недоступні, а також в предметних областях, що мають загальноприйняту усталену регламентну документацію. Також до цієї техніки відноситься і просто вивчення документів, що регламентують бізнес-процеси в предметній області замовника або в конкретній організації, що дозволяє отримати необхідні для кращого розуміння суті проекту знання.

Моделювання процесів та взаємодій

Може застосовуватися як до «Бізнес-процесів і взаємодій» (наприклад: «договір на закупівлю формується відділом закупівель, візується бухгалтерією і юридичним відділом ...»), так і до «технічних процесів і взаємодій» (наприклад: «платіжне доручення генерується модулем" Бухгалтерія ", шифрується модулем" Безпека "і передається на збереження в модуль" Сховище "»). Дана техніка вимагає високої кваліфікації фахівця з бізнес-аналізу, тому що пов'язана з обробкою великого обсягу складної (і часто погано структурованою) інформації.

Самостійний опис

Є не стільки технікою виявлення вимог, скільки технікою їх фіксації і формалізації. Дуже складно (і навіть неможливо!) намагатися самому «придумати вимоги за замовника», але в спокійній обстановці можна самостійно обробити зібрану інформацію і акуратно оформити її для подальшого обговорення і уточнення.

Властивості якісних вимог

Завершеність (completeness). Вимога є повною і закінченою з точки зору представлення в ній всієї необхідної інформації, нічого не пропущено з міркувань «це і так всім зрозуміло».

Типові проблеми з завершеністю:

- Відсутні нефункціональні складові вимоги або посилання на відповідні нефункціональні вимоги (наприклад: «паролі повинні зберігатися в зашифрованому вигляді»- який алгоритм шифрування?).
- Вказана лише частина деякого перерахування (наприклад: «експорт здійснюється в формати PDF, PNG і т.д. » - що ми повинні розуміти під "і т.д."?).
- Наведені посилання неоднозначні (наприклад: «див. Вище» замість «див. 123.45.b»).

Властивості якісних вимог

Атомарність, одиничність (atomicity). Вимога є атомарною, якщо її не можна розбити на окремі вимоги без втрати завершеності і вона описує одну і тільки одну ситуацію.

Типові проблеми з атомарністю:

- В одній вимозі, фактично, міститься кілька незалежних (Наприклад: «кнопка" Restart "не повинна відображатися при зупиненому сервісі, вікно "Log" має вміщувати не менше 20-ти записів про останні діях користувача »- тут для чогось в одному реченні описані абсолютно різні елементи інтерфейсу в абсолютно різних контекстах).
- Вимога допускає різночитання в силу граматичних особливостей мови (наприклад: «якщо користувач підтверджує замовлення і редагує замовлення або відкладає замовлення, повинен видаватися запит на оплату »- тут описані три різні випадки, і цю вимогу варто розбити на три окремих щоб уникнути плутанини). Таке порушення атомарності часто тягне за собою виникнення суперечливості.
- В одній вимозі об'єднано опис декількох незалежних ситуацій (наприклад: «коли користувач входить в систему, йому повинно відображатися вітання; коли користувач увійшов в систему, має відображатися ім'я користувача; коли користувач виходить з системи, має відображатися прощання »- всі ці три ситуації заслуговують того, щоб бути описаними окремими і куди більш детальними вимогами).

Властивості якісних вимог

Несуперечливість, послідовність (consistency). вимога не повинно містити внутрішніх протиріч і суперечностей іншим вимогам і документам.

Типові проблеми з несуперечливістю :

- Суперечності всередині однієї вимоги (наприклад: «після успішного входу в систему користувача, який не має права входити в систему ... » - тоді як він успішно увійшов до системи, якщо не мав такого права?)
- Протиріччя між двома і більше вимогами, між таблицею і текстом, малюнком і текстом, вимогою і прототипом і т.д. (Наприклад: «712.а Кнопка "Close" завжди повинна бути червоною » і « 36452.х Кнопка "Close" завжди повинна бути синьою »- так все ж червоною або синьою?)
- Використання невірної термінології або використання різних термінів для позначення одного і того ж об'єкта або явища (наприклад: «у разі, якщо розширення вікна становить менше 800х600 ... »- розширення є у екрану, у вікна є розмір).

Властивості якісних вимог

Недвозначність (unambiguousness , Clearness). вимога має бути описано без використання жаргону, неочевидних абревіатур і розпливчастих формулювань, має допускати тільки однозначне об'єктивне розуміння і бути атомарним в плані неможливості різного трактування поєднання окремих фраз. Типові проблеми з недвозначністю:

- Використання термінів або фраз, що допускають суб'єктивне тлумачення (Наприклад: «додаток повинен підтримувати передачу великих обсягів даних »- наскільки« великих »?) Ось лише невеликий перелік слів і виразів, які можна вважати вірними ознаками двозначності: адекватно (adequate), бути здатним (be able to), легко (easy), забезпечувати (provide for), як мінімум (as a minimum), бути здатним (be capable of), ефективно (effectively), своєчасно (timely), може бути застосовано (as applicable), якщо можливо (if possible), буде визначено пізніше (to be determined, TBD), у міру необхідності (as appropriate), якщо це доцільно (if practical), але не обмежуючись (but not limited to), бути здатне (capability of), мати можливість (capability to), нормально (normal), мінімізувати (Minimize), максимізувати (maximize), оптимізувати (optimize), швидко (Rapid), зручно (user-friendly), просто (simple), часто (often), зазвичай (usual), великий (large), гнучкий (flexible), стійкий (robust), за останнім словом техніки (state-of-the-art), покращений (improved), результативно (efficient). Ось перебільшений приклад вимоги, що звучить дуже красиво, але абсолютно нереалізованого і незрозумілого: «У разі необхідності оптимізації передачі великих файлів система повинна ефективно використовувати мінімум оперативної пам'яті, якщо це можливо ».
- Використання неочевидних або двозначних абревіатур без розшифровки (наприклад: «доступ до ФС здійснюється за допомогою системи прозорого шифрування »і« ФС надає можливість фіксувати повідомлення в їх поточному стані зі зберіганням історії всіх змін »- ФС тут позначає файлову систему? Точно? А не якийсь там «Фіксатор Сповіщень»?)
- Формулювання вимог з міркувань, що щось повинно бути всім очевидно (наприклад: «Система конвертує вхідний файл з формату PDF в вихідний файл формату PNG »- і при цьому автор вважає абсолютно очевидним, що імена файлів система отримує з командного рядка, а багатосторінковий PDF конвертується в кілька PNG-файлів, до імен яких додається «page-1», «page-2» і т.д.). Ця проблема перегукується з порушенням коректності.

Властивості якісних вимог

Здійснимість (feasibility). Вимога повинна бути мати технологічної здійсненною і реалізованою в рамках бюджету і термінів розробки проекту.

Типові проблеми з здійснимістю:

- Так звана «позолота» (gold plating) - вимоги, які вкрай довго і / або дорого реалізуються і при цьому практично не приносять користі для кінцевих користувачів (наприклад: «налаштування параметрів для підключення до бази даних повинне підтримувати розпізнавання символів з жестів, отриманих з пристроїв тривимірного введення»).
- Вимоги, що не можна технічно реалізувати на сучасному рівні розвитку технологій (наприклад: «аналіз договорів повинен виконуватися із застосуванням штучного інтелекту, який буде виносити однозначний коректний висновок про ступінь вигоди від укладення договору »).
- В принципі неможливі вимоги (наприклад: «система пошуку повинна заздалегідь передбачати всі можливі варіанти пошукових запитів і кешувати їх результати »).

Властивості якісних вимог

Обов'язковість, потрібність (obligatoriness) і актуальність (up-to-date).

Якщо вимога не є обов'язковою до реалізації, воно повинно бути просто виключено з набору вимог. Якщо вимога потрібне, але «не дуже важливе», для вказівки цього факту використовується вказівка пріоритету (див. «Проранжованість по ...»). Також виключені (або перероблені) повинні бути вимоги, що втратили актуальність.

Типові проблеми з обов'язковістю і актуальністю:

- Вимога була додана «про всяк випадок», хоча реальної потреби в ній немає і не буде
- Вимозі виставлені невірні значення пріоритету за критеріями важливості і / або терміновості.
- Вимога застаріла, але не була перероблена або видалена.

Властивості якісних вимог

Відстежуваність (traceability). Відстежуваністьбуває вертикальною (vertical traceability) і горизонтальною (horizontal traceability). вертикальна дозволяє співвідносити між собою вимоги на різних рівнях вимог, горизонтальна дозволяє співвідносити вимога з тест-планом, тест-кейсами, архітектурними рішеннями і т.д. Для забезпечення відстежуваності часто використовуються спеціальні інструменти з управління вимогами (requirements management tool) і / або матриці простежуваності (traceability matrix.

Типові проблеми з відстежуваністю:

- Вимоги не пронумеровані, не структуровані, не мають змісту, не мають працюючих перехресних посилань.
- При розробці вимог не були використані інструменти і техніки управління вимогами.
- Набір вимог неповний, носить уривчастий характер з явними «пробілами».

Властивості якісних вимог

Модифікованість(modifiability). Це властивість характеризує простоту внесення змін в окремі вимоги і в набір вимог. Можна говорити про наявність модифікованості в тому випадку, якщо при доопрацюванні вимог шукану інформацію легко знайти, а її зміна не призводить до порушення інших описаних в цьому переліку властивостей.

Типові проблеми з модифікованістю:

- Вимоги неатомарні і невідстежувані, а тому їх зміна з високою ймовірністю породжує суперечливість.
- Вимоги спочатку суперечливі. У такій ситуації внесення змін (не пов'язаних з усуненням суперечливості) тільки погіршує ситуацію, збільшуючи суперечливість і знижуючи відстежуваність.
- Вимоги представлені в незручній для обробки формі (наприклад, не використані інструменти управління вимогами, і в підсумку команді доводиться працювати з десятками величезних текстових документів).

Властивості якісних вимог

Проранжованість за такими характеристиками як важливість, стабільність, терміновість (ranked for importance, stability, priority). Важливість характеризує залежність успіху проекту від успіху реалізації вимоги. Стабільність характеризує ймовірність того, що в доступному для огляду майбутньому в вимога не буде внесено ніяких змін. Терміновість визначає розподіл в часі зусиль проектної команди по реалізації того чи іншого вимоги.

Типові проблеми з проранжованістю складаються в її відсутності або неправильній реалізації і призводять до таких наслідків.

- Проблеми з проранжованістю за важливістю підвищують ризик невірного розподілу зусиль проектної команди, спрямування зусиль на другорядні завдання і кінцевого провалу проекту через нездатність продукту виконувати ключові завдання з дотриманням ключових умов.
- Проблеми з проранжованістю по стабільності підвищують ризик виконання безглуздої роботи по вдосконаленню, реалізації та тестування вимог, які в самий найближчий час можуть зазнати кардинальні зміни (аж до повної втрати актуальності).
- Проблеми з проранжованістю по терміновості підвищують ризик порушення бажаної замовником послідовності реалізації функціональності і введення цієї функціональності в експлуатацію.

Властивості якісних вимог

Коректність (correctness) і перевірюваність (verifiability). фактично ці властивості випливають з дотримання всіх перерахованих вище (або можна сказати, що вони не виконуються, якщо порушено хоча б одне з перерахованих вище). На додаток можна відзначити, що перевірюваність має на увазі можливість створення об'єктивного тест-кейса (тест-кейсів), що однозначно показують, що вимога реалізовано вірно і поведінка застосування відповідає вимозі.

До типових проблем з коректністю також можна віднести:

- помилки (особливо небезпечні помилки в абревіатурах, що перетворюють одну абревіатуро в іншу; такі помилки вкрай складно помітити);
- наявність неаргументованих вимог до дизайну та архітектури;
- погане оформлення тексту і супутньої графічної інформації, граматичні, пунктуаційні та інші помилки в тексті;
- невірний рівень деталізації (наприклад, занадто глибока деталізація вимоги на рівні бізнес-вимог або недостатня деталізація на рівні вимог до продукту);
- вимоги до користувача, а не до додатка (наприклад: «користувач повинен бути в змозі відправити повідомлення »- на жаль, ми не можемо впливати на стан користувача).

Функціональне тестування ПЗ та основи тест-дизайну

Лекція 2

Розробка тестів та тестових сценаріїв

Визначення

Чек-ліст (checklist) - набір ідей [для написання тест-кейсів]. Це документ, який описує, що має бути протестованим. При цьому чек-ліст може бути абсолютно різного рівня деталізації. деталізація буде залежати від вимог до звітності, рівня знань продукту і складності продукту.

Чек-ліст дозволяє не забути про важливі тести, фіксувати результати своєї роботи і відслідковувати статистику про статус програмного продукту

Визначення

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

Визначення

Набір тест-кейсів (test suite, test set) - сукупність тест-кейсів, для тестування компоненту або системи, обрані таким чином, що стан системи при завершенні одного тесту найчастіше є станом (pre-condition) для початку іншого.

Завдання Гленфорда Майерса про трикутник

Програма на С (32-бітний компілятор під Win32) проводить читання з командного рядка трьох цілих чисел, які інтерпретуються як довжини сторін трикутника. Далі програма виводить в консоль повідомлення про те, чи є трикутник нерівностороннім, рівнобедреним або рівностороннім.

Висновок

Без записаного чек-листа вже через кілька хвилин ідеї починають дублюватися, губитися, спотворюватися і т.д. Починайте з простих очевидних тест-кейсів, які перевіряють працездатність основних функцій програми.

Пам'ятайте про рекомендовану послідовності розробки та виконання тест кейсів:

- Прості позитивні.
- Прості негативні.
- Складні позитивні.
- Складні негативні.

Якщо залишається час, займайтеся дослідним тестуванням.

Стандартний вигляд тест кейсу

Project Name:

Test Case Template

Test Case ID: Fun_10
Test Designed by: <Name>

Test Priority (Low/Medium/High): Med
Test Designed date: <Date>

Module Name: Google login screen
Test Executed by: <Name>

Test Title: Verify login with valid username and password
Test Execution date: <Date>

Description: Test the Google login page

Pre-conditions: User has valid username and password
Dependencies:

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Navigate to login page	User= example@gmail.com	User should be able to login	User is navigated to	Pass	
2	Provide valid username	Password: 1234		dashboard with successful		
3	Provide valid password			login		
4	Click on Login button					

Post-conditions:
User is validated with database and successfully login to account. The account session details are logged in database.

Тест кейс. Варіант 2

Test Scenario ID	Login-1	Test Case ID	Login-1A
Test Case Description	Login – Positive test case	Test Priority	High
Pre-Requisite	A valid user account	Post-Requisite	NA

Test Execution Steps:

S.No	Action	Inputs	Expected Output	Actual Output	Test Browser	Test Result	Test Comments
1	Launch application	https://www.facebook.com/	Facebook home	Facebook home	IE-11	Pass	[Priya 10/17/2017 11:44 AM]: Launch successful
2	Enter correct Email & Password and hit login button	Email id : test@xyz.com Password: ******	Login success	Login success	IE-11	Pass	[Priya 10/17/2017 11:45 AM]: Login successful

Поля тест-кейсу

Ідентифікатор

- Унікальний.
- Осмислений (якщо дозволяє ПЗ).

Пріоритет

- Показує важливість тест-кейса.
- 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.
- Повинен корелювати з:
  - о важливістю вимоги;
  - о потенційної важливістю дефекту, на пошук якого спрямований тест кейс.

Поля тест-кейсу

Модуль і підмодуль програми

• Вказують на частини програми, до яких відноситься тест-кейс і дозволяють краще зрозуміти його мету.

• Модуль і підмодуль програми - це НЕ дії, це саме структурні частини, «шматки» програми. Порівняйте (на прикладі людини): «дихальна система, легені » - це модуль і підмодуль, а« дихання »,« сопіння »,« чхання » - ні; «Голова, мозок» - це модуль і підмодуль, а «кивання», «думання» - ні.

Поля тест-кейсу

Назва (суть) тест-кейса

- Покликана спростити швидке розуміння основної ідеї тест-кейса без звернення до його іншим атрибутам. Тобто повинна коротко відображати цілі, що перевіряє тест кейс.
- Саме це поле є найбільш інформативним при перегляді списку тест-кейсів.
- Назва завжди має бути! У кожного тест-кейса! Ні за яких умов категорично не допускається наявність тест-кейсів без назв!

Поля тест-кейсу

Вихідні дані, приготування

• Дозволяють описати все те, що повинно бути підготовлено до початку виконання тест-кейса, наприклад, стан бази даних, стан файлової системи і її об'єктів і т.д. Як варіант передумови можуть бути виконання набору або одного тест-кейса, який призводить додаток в певний стан.

• Якщо говорити, наприклад, про підготовку навколишнього середовища, то все, що описується в цьому полі, готується БЕЗ використання тестованого додатку. Відповідно, якщо тут виникають проблеми, не можна писати звіт про дефект, який був виявлений в процесі проходження даного тест кейса (якщо причиною став стан БД, теж потрібно писати про дефект, але в БД). Аналогічно, якщо умовою появи дефекту є тест-кейс з передумови (pre-condition), то потрібно писати звіт, саме до того тест-кейсу

Поля тест-кейсу

Кроки тест-кейса і очікувані результати

- Кроки тест-кейса описують послідовність дій, які необхідно реалізувати в процесі виконання тест-кейса.
- Очікувані результати по кожному кроку тест-кейса описують реакцію додатку на дії, описані в полі «кроки тест-кейса».
- Номер кроку відповідає номеру результату.

Поля тест-кейсу

Мова написання тест-кейсів

- Використовуйте «open», «paste», «click». В українській мові використовуйте безособову форму дієслова (наприклад, «відкрити» замість «відкрийте»).
- Описуйте поведінку системи, яку можна об'єктивно перевірити: «з'являється вікно ...», «Додаток закривається».
- Використовуйте простий технічний стиль.
- ОБОВ'ЯЗКОВО вказуйте ТОЧНІ назви всіх елементів програми.
- Не пояснюйте базові поняття роботи з ОС.
- Стандарти написання тест-кейсів дуже важливі в проектах, в яких їх багато і багато тестувальників. Повинен бути Single writing style, іноді аж до того що кнопки пишуться в подвійних лапках, дропдауни в одинарних, і тд.

Властивості якісних тест-кейсів

Баланс між специфічністю та узагальненням

У разі зайвої специфічності:

• при повторних виконаннях тест-кейсу завжди будуть виконуватися строго одні й ті ж дії, що знижує ймовірність виявити помилку;

• зростає час створення і підтримки тест-кейса.

У разі сильного узагальнення:

• тест-кейс складний для початківців тестувальників;

• тест-кейс цілком може залишитися невиконаним по ряду об'єктивних і суб'єктивних причин.

Додавання A і B

1. В поле A ввести коректне ціле число.
2. В поле B ввести коректне ціле число.
3. Натиснути кнопку «Додати»
4. Перевірити значення поля C
5. Повторити кроки 1-4 для значень: 0, max і min допустимих значень

4. Значення поля C повинне дорівнювати сумі A і B

- Тут ми не прив'язані до конкретних значень.
- Ми знаємо, як перевірити результат.
- Ми скорочуємо час написання і підтримки тест-кейса посиланням на кроки 1-4.
- Ми перерахували значення, що мають для нас особливий інтерес.

Властивості якісних тест-кейсів

Баланс між простотою і складністю

Простий тест-кейс оперує одним об'єктом (або в ньому явно видно головний об'єкт), а також містить невелику кількість тривіальних дій

Складний тест-кейс оперує кількома рівноправними об'єктами і містить багато нетривіальних дій. Простота і складність самі по собі нічим не погані. Проблеми починаються з зайвої простоти і зайвої складності.

Переваги простих тест-кейсів:

- Їх легко виконувати.
- Вони зрозумілі новачкам.
- Вони спрощують діагностику помилки.
- Вони роблять наявність помилки очевидною.

Переваги складних тест-кейсів:

- Більше шансів щось зламати.
- Користувачі, як правило, використовують складні сценарії.
- Програмісти самі рідко перевіряють такі варіанти.

Слід поступово підвищувати складність тест-кейсів.

Властивості якісних тест-кейсів

Незалежність або обґрунтована зв'язаність

Незалежні тест-кейси не посилаються на жодні інші.

Пов'язані тест-кейси явно або неявно (в рамках набору) посилаються на інші (Як правило, на попередній).

Переваги незалежних тест-кейсів:

- Легко і просто виконувати.
- Можуть працювати навіть після збою програми на інших тест-кейсах.
- Можна групувати будь-яким чином і виконувати в будь-якому порядку.

Переваги пов'язаних тест-кейсів і наборів тест-кейсів:

- Імітують роботу реальних користувачів.
- Наступний в наборі тест-кейс використовує дані і стан додатку, підготовлені попереднім. Тобто кожен наступний рівень пов'язаних тест кейсів копає глибше (логічний зв'язок з рівнями тестування).

Хороший тест-кейс:

- Володіє високою ймовірністю виявлення помилки.
- Послідовний у досягненні мети.
- Не містить зайвих (безглуздих) дій.
- Не є надмірним по відношенню до інших тест-кейсів.
- Робить виявлену помилку очевидною.
- Виконує якісь цікаві дії (особливо розглядаючи негативні кейси, але це не може бути застосовано до димового тестування).
- Зрозумілий людині, що вперше прийшла на проект.

Завдання

Як відправити на сервер (за протоколом HTTP) файл з іменем «% ^ ## 76 / // \ ^^ []: .jpg »?

Цей файл ТОЧНО не можна створити в жодній файловій системі. але відправити файл з таким іменем можна. Як?

Процес розробки тестів

1. Починайте якомога раніше, ще до виходу першого білду.
2. Розбивайте додаток на окремі частини / модулі.
3. Для кожної області / модуля пишіть чек-лист.
4. Пишіть питання, уточнюйте деталі, додавайте «косметику», використовуйте copy- paste.
5. Отримайте рецензію колег-тестувальників, розробників, замовників.
6. Обновляйте тест-кейси, як тільки виявили помилку або змінилася функціональність.

Набори тест-кейсів

Набір тест-кейсів (test case suite, test suite, test set) - сукупність тест-кейсів, обраних з деякою загальною метою або за деякою спільною ознакою.

Іноді в такий сукупності результати завершення одного тест-кейса стають вхідним станом додатку для наступного тест-кейса.

Вільні набори - порядок виконання тест-кейсів не важливий.

Послідовні набори - порядок виконання тест-кейсів важливий.

Переваги вільних наборів:

• Тест-кейси можна виконувати в 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 в наборі тест-кейс в якості вхідного стану додатку 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» впливу на додаток.

Функціональне тестування ПЗ та основи тест-дизайну

Лекція 3

Види і напрями тестування

Спрощена класифікація тестування

Тестування можна класифікувати по дуже великій кількості ознак, і практично в кожній серйозній книзі про тестування автор показує свій (безумовно має право на існування) погляд на це питання.

Відповідний матеріал досить об'ємний і складний, а глибоке розуміння кожного пункту в класифікації вимагає певного досвіду.

За запуском коду на виконання:

- Статичне тестування - без запуску.
- Динамічне тестування - з запуском.

За доступом до коду та архітектури програми:

- Метод білого ящика - доступ до коду є.
- Метод чорного ящика - доступу до коду немає.
- Метод сірого ящика - до частини коду доступ є, до частини - немає.

За ступенем автоматизації:

- Ручне тестування - тест-кейси виконує людина.
- Автоматизоване тестування - тест-кейси частково або повністю виконує спеціальний інструментальний засіб.

За рівнем деталізації додатку (за рівнем тестування):

- Модульне (компонентне) тестування - перевіряються окремі невеликі частини програми.
- Інтеграційне тестування - перевіряється взаємодія між декількома частинами програми.
- Системне тестування - додаток перевіряється як єдине ціле.

За (зменшенням) ступенем важливості тестованих функцій (за рівнем функціонального тестування):

- Димове тестування - перевірка найважливішою, самої ключової функціональності, непрацездатність якої робить безглуздою саму ідею використання програми.
- Тестування критичного шляху - перевірка функціональності, використовуваної типовими користувачами в типовій повсякденній діяльності.
- Розширене тестування - перевірка всієї (решти) функціональності, заявленої в вимогах.

За принципами роботи з додатком:

- Позитивне тестування - всі дії з додатком виконуються строго по інструкції без жодних неприпустимих дій, некоректних даних і т.д. Можна образно сказати, що додаток досліджується в «тепличних умовах».
- Негативне тестування - в роботі з додатком виконуються (Некоректні) операції та використовуються дані, що потенційно призводять до помилок (класика жанру – ділення на нуль).

Увага! Дуже часта помилка! Негативні тести НЕ припускають виникнення в додатку помилки. навпаки - вони припускають, що додаток вірно працює і навіть в критичній ситуації поведе себе правильним чином (в прикладі з діленням на нуль, наприклад, відобразить повідомлення «Ділити на нуль заборонено»).

Якщо вас цікавить якась «еталонна класифікація», то ... її не існує. Можна сказати, що в матеріалах ISTQB наведено найбільш узагальнений і загальноприйнятий погляд на це питання, але і там немає єдиної схеми, яка об'єднувала б усі варіанти класифікації.

Так що, якщо вас просять розповісти про класифікацію тестування, варто уточнити, згідно з яким автору або джерела запитувач очікує почути вашу відповідь.

За запуском коду на виконання:

Далеко не всяке тестування передбачає взаємодію з працюючим додатком. Тому в рамках даної класифікації виділяють:

Статичне тестування (static testing) - тестування без запуску коду на виконання. В рамках цього підходу тестування можуть потрапити під вплив:

- Документи (вимоги, тест-кейси, опису архітектури додатку, схеми баз даних і т.д.).
- Графічні прототипи (наприклад, ескізи призначеного для користувача інтерфейсу).
- Код додатку (що часто виконується самими програмістами в рамках аудиту коду (code review), що є специфічною варіацією взаємного перегляду в застосуванні до вихідного коду). Код додатку також можна перевіряти з використанням технік тестування на основі структур коду.
- Параметри (налаштування) середовища виконання програми.
- Підготовлені тестові дані.

За запуском коду на виконання:

Динамічне тестування (dynamic testing) - тестування з запуском коду на виконання. Запускатися на виконання може як код всієї програми цілком (системне тестування), так і код декількох взаємопов'язаних частин (інтеграційне тестування), окремих частин (модульне або компонентне тестування) і навіть окремі ділянки коду. Основна ідея цього виду тестування полягає в тому, що перевіряється реальна поведінку (частини) додатку.

За доступом до коду та архітектури програми:

Метод білого ящика (white box testing, Open box testing, clear box testing,glass box testing) - у тестувальника є доступ до внутрішньої структурі та коду додатку, а також є достатньо знань для розуміння побаченого. Виділяють навіть супутню до тестування за методом білого ящика глобальну техніку - тестування на основі дизайну (design-based testing).

Деякі автори схильні жорстко пов'язувати цей метод зі статичним тестуванням, але ніщо не заважає тестувальнику запустити код на виконання і при цьому періодично звертатися до самого коду (а модульне тестування і зовсім передбачає запуск коду на виконання і при цьому роботу саме з кодом, а не з «Додатком повністю»).

За доступом до коду та архітектури програми:

Метод чорного ящика (black box testing, Closed box testing, specification based testing) - у тестувальника або немає доступу до внутрішньої структури та коду програми, або недостатньо знань для їх розуміння, або він свідомо не звертається до них у процесі тестування. При цьому абсолютна більшість перерахованих на схемах видів тестування працюють за методом чорного ящика, ідею якого в альтернативному визначенні можна сформулювати так: тестувальник впливає на додаток (і перевіряє реакцію) тим же способом, яким при реальній експлуатації додатку на нього впливали б користувачі або інші додатки. В рамках тестування за методом чорного ящика основною інформацією для створення тест-кейсів виступає документація (Особливо - вимоги (requirements-based testing)) і загальний здоровий глузд (для випадків, коли поведінка додатку в деякій ситуації не регламентована явно; іноді це називають «тестуванням на основі неявних вимог», але канонічного визначення у цього підходу немає).

За доступом до коду та архітектури програми:

Метод сірого ящика (gray box testing) - комбінація методів білого ящика і чорного ящика, яка полягає в тому, що до частини коду і архітектури у тестувальника доступ є, а до частини - немає. Це вкрай рідкісний випадок: зазвичай говорять про методи білого або чорного ящика в застосуванні до тих чи інших частин додатку, при цьому розуміючи, що «додаток повністю» тестується за методом сірого ящика.

За ступенем автоматизації:

Ручне тестування (manual testing) - тестування, в якому тест кейси виконуються людиною вручну без використання засобів автоматизації. Незважаючи на те що це звучить дуже просто, від тестувальника в ті чи інші моменти часу потрібні такі якості, як терплячість, спостережливість, креативність, вміння ставити нестандартні експерименти, а також уміння бачити і розуміти, що відбувається «всередині системи», тобто як зовнішні впливи на додаток трансформуються в його внутрішні процеси.

За ступенем автоматизації:

Автоматизоване тестування (automated testing, test automation) - набір технік, підходів і інструментальних засобів, що дозволяє виключити людину з виконання деяких завдань в процесі тестування.

Тест-кейси частково або повністю виконує спеціальний інструментальне засіб, однак розробка тест-кейсів, підготовка даних, оцінка результатів виконання, написання звітів про виявлені дефекти – все це і багато іншого, як і раніше робить людина

За рівнем деталізації додатку (за рівнем тестування):

Модульне (компонентне) тестування (unit testing, module testing, component testing) направлене на перевірку окремих невеликих частин програми, які (як правило) можна досліджувати ізольовано від інших подібних частин. При виконанні даного тестування можуть перевірятися окремі функції або методи класів, самі класи, взаємодія класів, невеликі бібліотеки, окремі частини програми. Часто даний вид тестування реалізується з використанням спеціальних технологій і інструментальних засобів автоматизації тестування що значно спрощують і прискорюють розробку відповідних тест-кейсів.

За рівнем деталізації додатку (за рівнем тестування):

Інтеграційне тестування (integration testing , Component integration testing, Pairwise integration testing, System integration testing, incremental testing, Interface testing, Thread testing) направлено на перевірку взаємодії між декількома частинами додатку (кожна з яких, в свою чергу, перевірена окремо на стадії модульного тестування). Нажаль, навіть якщо ми працюємо з дуже якісними окремими компонентами, «на стику» їх взаємодії часто виникають проблеми.

Саме ці проблеми і виявляє інтеграційне тестування.

За рівнем деталізації додатку (за рівнем тестування):

Системне тестування (system testing) направлено на перевірку всього додатку як єдиного цілого, складеного з частин, перевірених на двох попередніх стадіях. Тут не тільки виявляються дефекти «на стиках» компонентів, але і з'являється можливість повноцінно взаємодіяти з додатком з точки зору кінцевого користувача, застосовуючи безліч інших видів тестування, перерахованих в цьому розділі.

За (зменшенням) ступенем важливості тестованих функцій (за рівнем функціонального тестування):

Димове тестування - перевірка найважливішою, самої ключової функціональності, непрацездатність якої робить безглуздою саму ідею використання програми.

Димове тестування проводиться після виходу нового білду, щоб визначити загальний рівень якості додатку і прийняти рішення про (не) доцільність виконання тестування критичного шляху і розширеного тестування. Оскільки тест-кейсів на рівні димового тестування відносно небагато, а самі вони досить прості, але при цьому дуже часто повторюються, вони є хорошими кандидатами на автоматизацію. В зв'язку з високою важливістю тест-кейсів на даному рівні порогове значення метрики їх проходження часто виставляється рівним 100% або близьким до 100%.

За (зменшенням) ступенем важливості тестованих функцій (за рівнем функціонального тестування):

Тестування критичного шляху (critical path test) направлене на дослідження функціональності, використовуваної типовими користувачами в типовій повсякденній діяльності.

Існує безліч користувачів, які найчастіше використовують деяку підмножину функцій програм. Саме ці функції і потрібно перевірити, як тільки ми переконалися, що додаток «в принципі працює »(димовий тест пройшов успішно). Якщо з якихось причин програма не виконує ці функції або виконує їх некоректно, дуже багато користувачів не зможуть досягти безлічі своїх цілей. Граничне значення метрики успішного проходження «тесту критичного шляху» вже трохи нижче, ніж в димовому тестуванні, але все одно досить високе (Як правило, близько 70-80-90% - в залежності від суті проекту).

За (зменшенням) ступенем важливості тестованих функцій (за рівнем функціонального тестування):

Розширене тестування (extended test) направлене на дослідження всієї заявленої в вимогах функціональності - навіть тієї, яка низько проранжована за ступенем важливості. При цьому тут також враховується, яка функціональність є більш важливою, а яка - менш важливою. Але при наявності достатньої кількості часу і інших ресурсів тест-кейси цього рівня можуть торкнутися навіть найбільш фонових вимог.

Ще одним напрямком дослідження в рамках даного тестування є нетипові, малоймовірні, екзотичні випадки і сценарії використання функцій і властивостей додатку, розглянутих на попередніх рівнях. Граничне значення метрики успішного проходження розширеного тестування істотно нижче, ніж в тестуванні критичного шляху (Іноді можна побачити навіть значення в діапазоні 30-50%, тому що переважна більшість знайдених тут дефектів не становить загрози для успішного використання програми більшістю користувачів).

За принципами роботи з додатком:

Позитивне тестування (positive testing) направлене на дослідження додатку в ситуації, коли всі дії виконуються строго по інструкції без жодних помилок, відхилень, введення невірних даних і т.д. Якщо позитивні тест-кейси завершуються помилками, це тривожна ознака - додаток не працює належним чином навіть в ідеальних умовах (і можна припустити, що в неідеальних умовах він працює ще гірше).

Для прискорення тестування кілька позитивних тест-кейсів можна об'єднувати (наприклад, перед відправкою заповнити всі поля форми вірними значеннями) - іноді це може ускладнити діагностику помилки, але суттєва економія часу компенсує цей ризик.

За принципами роботи з додатком:

Негативне тестування (negative testing, Invalid testing) – направлене на дослідження роботи програми в ситуаціях, коли з нею виконуються (Некоректні) операції та / або використовуються дані, що потенційно призводять до помилок. Оскільки в реальному житті таких ситуацій значно більше (користувачі допускають помилки, зловмисники свідомо «ламають» додаток, в середовищі роботи додатку виникають проблеми і т.д.), негативних тест-кейсів виявляється значно більше, ніж позитивних (іноді - в рази або навіть на порядки). На відміну від позитивних негативні тест-кейси не варто об'єднувати, тому що подібне рішення може привести до невірного трактування поведінки додатку і пропуску (невиявлення) дефектів.

Класифікація за природою додатку

Даний вид класифікації є штучним, оскільки «всередині» мова буде йти про одні й ті ж види тестування, що відрізняються в даному

контексті лише концентрацією на відповідних функціях і особливостях застосування, використанням специфічних інструментів та окремих технік.

Тестування веб-додатків (web-applications testing) пов'язане з інтенсивною діяльністю в області тестування сумісності (Особливо - крос-браузерного тестування), тестування продуктивності, автоматизації тестування з використанням широкого спектра інструментальних засобів.

Тестування мобільних додатків (mobile applications testing) також вимагає підвищеної уваги до тестування сумісності, оптимізації продуктивності (В тому числі клієнтської частини з точки зору зниження енергоспоживання), автоматизації тестування із застосуванням емуляторів мобільних пристроїв.

Тестування настільних додатків (desktop applications testing) є найбільш класичним серед всіх перерахованих в даній класифікації, і його особливості залежать від предметної області додатку, нюансів архітектури, ключових показників якості і т.д.

Цю класифікацію можна продовжувати дуже довго. Наприклад, можна окремо розглядати тестування консольних додатків (console applications testing) і додатків з графічним інтерфейсом (GUI-applications testing), серверних додатків (server applications testing) і клієнтських додатків (client applications testing) і т.д.

Класифікація за цілями і завданнями

Позитивне тестування (розглянуто раніше).

Негативне тестування (розглянуто раніше).

Функціональне тестування (functional testing) - вид тестування, спрямований на перевірку коректності роботи функціональності додатку (коректність реалізації функціональних вимог). Часто функціональне тестування асоціюють з тестуванням по методу чорного ящика , однак і за методом білого ящика цілком можна перевіряти коректність реалізації функціональності.

Нефункціональне тестування (non-functional testing) - вид тестування, спрямований на перевірку не функціональних особливостей програми (коректність реалізації функціональних вимог), Таких як зручність використання, сумісність, продуктивність, безпеку і т.д.

Класифікація за цілями і завданнями

Інсталяційне тестування (installation testing, installability testing) - тестування, спрямоване на виявлення дефектів, що впливають на перебіг стадії інсталяції (установки) додатку. У загальному випадку таке тестування перевіряє безліч сценаріїв і аспектів роботи інсталятора в таких ситуаціях, як:

- нове середовище виконання, в якому додаток раніше не було інстальовано;
- оновлення існуючої версії («апгрейд»);
- зміна поточної версії на більш стару («даунгрейд»);
- повторна установка програми з метою усунення проблем, що виникли («перевстановлення»);
- повторний запуск інсталяції після помилки, що призвела до неможливості продовження інсталяції;
- видалення програми;
- установка нового додатку з сімейства додатків;
- автоматична інсталяція без участі користувача.

Класифікація за цілями і завданнями

Регресійне тестування (regression testing) - тестування, спрямоване на перевірку того факту, що в раніше робочій функціональності не з'явилися помилки, викликані змінами в додатку або середовищі його функціонування. Фредерік Брукс у своїй книзі «Міфічний людино-місяць» писав: «Фундаментальна проблема при супроводі програм полягає в тому, що виправлення однієї помилки з великою ймовірністю (20-50%) тягне за собою появу нової». Тому регресійне тестування є невід'ємним інструментом забезпечення якості і активно використовується практично в будь-якому проекті.

Повторне тестування (re-testing, confirmation testing) – виконання тест-кейсів, які раніше виявили дефекти, з метою підтвердження усунення дефектів. Фактично цей вид тестування зводиться до дій на фінальній стадії життєвого циклу звіту про дефект , спрямованим на те, щоб перевести дефект в стан «перевірений» і «закритий».

Класифікація за цілями і завданнями

Приймальне тестування (acceptance testing) - формалізоване тестування, спрямоване на перевірку додатку з точки зору кінцевого користувача / замовника і винесення рішення про те, чи приймає замовник роботу у виконавця (проектної команди). Можна виділити наступні підвиди приймального тестування (хоча згадують їх вкрай рідко, обмежуючись в основному загальним терміном «приймальне тестування»):

- Виробниче приймальне тестування (factory acceptance testing) - виконується проектною командою дослідження повноти і якості реалізації програми з точки зору його готовності до передачі замовнику. Цей вид тестування часто розглядається як синонім альфа-тестування.
- Операційне приймальне тестування (operational acceptance testing, Production acceptance testing) - операційне тестування , що виконується з точки зору виконання інсталяції, споживання додатком ресурсів, сумісності з програмної і апаратної платформою і т.д.
- Підсумкове приймальне тестування (site acceptance testing) - тестування кінцевими користувачами (представниками замовника) додатку в реальних умовах експлуатації з метою винесення рішення про те, чи потребує додаток доробок або може бути прийнятий в експлуатацію в поточному вигляді.

Класифікація за цілями і завданнями

Операційне тестування (operational testing) - тестування, що проводиться в реальному або наближеному до реального операційному середовищі (Operational environment), що включає операційну систему, системи управління базами даних, сервери додатків, веб-сервери, апаратне забезпечення і т.д.

Тестування зручності використання (usability testing) - тестування, спрямоване на дослідження того, наскільки кінцевому користувачеві зрозуміло, як працювати з продуктом (understandability, learnability, operability), а також на те, наскільки йому подобається використовувати продукт (Attractiveness). І це не обмовка - дуже часто успіх продукту залежить саме від емоцій, які він викликає у користувачів. для ефективного проведення цього виду тестування потрібно реалізувати досить серйозні дослідження із залученням кінцевих користувачів, проведенням маркетингових досліджень і т.д.

Класифікація за цілями і завданнями

Тестування доступності (accessibility testing) - тестування, спрямоване на дослідження придатності продукту до використання людьми з обмеженими можливостями (слабким зором і т.д.)

Тестування інтерфейсу (interface testing) - тестування, спрямоване на перевірку інтерфейсів додатків або їхніх компонентів. За визначенням ISTQB-глосарію цей вид тестування відноситься до інтеграційного тестування, і це цілком справедливо для таких його варіацій як тестування інтерфейсу прикладного програмування (API testing) і інтерфейсу командного рядка (CLI testing), хоча воно може виступати і як різновид тестування користувальницького інтерфейсу, якщо через командний рядок з додатком взаємодіє користувач, а не інший додаток. Однак багато джерел пропонують включити до складу тестування інтерфейсу і тестування безпосередньо інтерфейсу користувача (GUI testing).

Класифікація за цілями і завданнями

Тестування безпеки (security testing) - тестування, спрямоване на перевірку здатності додатки протистояти зловмисним спробам отримання доступу до даних або функцій, права на доступ до яких у зловмисника немає.

Тестування інтернаціоналізації (internationalization testing, globalization testing, localizability testing) - тестування, спрямоване на перевірку готовності продукту до роботи з використанням різних мов і з урахуванням різних національних і культурних особливостей. Цей вид тестування не має на увазі перевірки якості відповідної адаптації (цим займається тестування локалізації, див. Наступний пункт), воно сфокусовано саме на перевірці можливості такої адаптації (наприклад: що буде, якщо відкрити файл з ієрогліфом в імені; як буде працювати інтерфейс, якщо все перевести на японський; чи може додаток шукати дані в тексті на корейському і т.д.)

Practice

URL: http://prestashop.qatestlab.com.ua/uk/

1. Write check-list (from 20 to 30 items);
2. Choose few items and write test cases (from 10 to 15 items) about them (use correct form with all required fields);

Функціональне тестування ПЗ та основи тест-дизайну

Лекція 4

Звіти про дефекти

Термінологія

Дефект - розбіжність очікуваного і фактичного результату.

Очікуваний результат - поведінка системи, описана в вимогах.

Фактичний результат - поведінка системи, що спостерігається в процесі тестування.

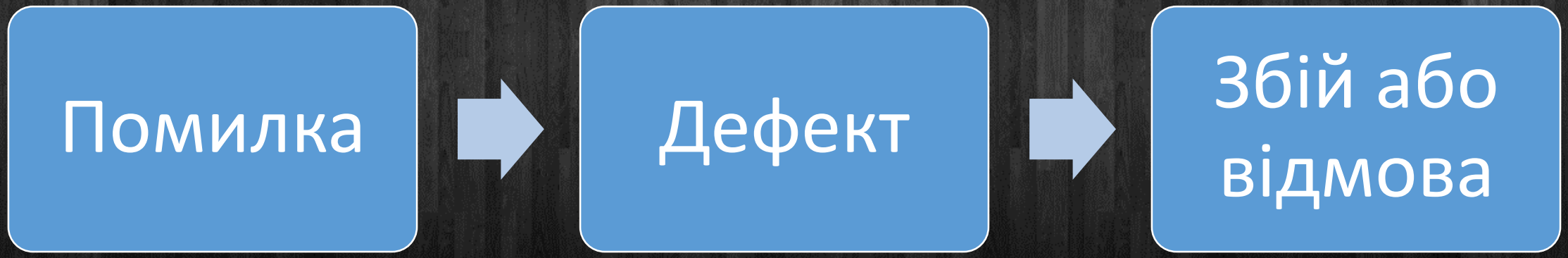
Розширений погляд на термінологію, що описує проблеми

Розберемося з широким спектром синонімів, якими позначають проблеми з програмними продуктами та іншими артефактами і процесами, супутніми їх розробці.

У сіллабусі ISTQB написано, що людина робить помилки, які призводять до виникнення дефектів в коді, які, в свою чергу, призводять до збоїв і відмов додатку (проте збої і відмови можуть виникати і через зовнішні умови, такі як електромагнітний вплив на обладнання і т.д.)

Таким чином, спрощено можна зобразити наступну схему:

Помилки, дефекти, збої та відмови



Помилка (error, mistake) - дія людини, що приводить 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 »і т.п.) Більш 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 (defect, bug, problem, fault) - недолік в компоненті або системі, здатний привести до ситуації збою або відмови. Цей термін також розуміють досить широко, кажучи про дефекти в документації, настройках, вхідних даних і т.д. Чому глава називається саме «звіти про дефекти»? Тому що цей термін як раз стоїть 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 в цьому напрямку є саме опис дефекту.

Збій (interruption) або відмова (failure) - відхилення поведінки системи від очікуваного.

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

Дефект - відхилення (deviation) фактичного результату (actual result) від очікувань спостерігача (expected result), сформованих на основі вимог, специфікацій, іншої документації або досвіду і здорового глузду.

Звідси логічно випливає, що дефекти можуть зустрічатися не тільки в коді додатку, але і в будь-якій документації, в архітектурі і дизайні, в настройках тестованої програми або тестового оточення - де завгодно.

Звіт про дефект і його життєвий цикл

Як було сказано в попередньому розділі, при виявленні дефекту тестувальник створює звіт про дефект.

Звіт про дефект (defect report317) - документ, що описує і пріорітізірующій виявлений дефект, а також сприяє його усуненню.

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

- надати інформацію про проблему - повідомити проектну команду і інших зацікавлених осіб про наявність проблеми, описати суть проблеми;
- пріоритезувати проблему - визначити ступінь небезпеки проблеми для проекту і бажані терміни її усунення;
- сприяти усуненню проблеми - якісний звіт про дефекти не тільки надає всі необхідні подробиці для розуміння суті що сталося, але також може містити аналіз причин виникнення проблеми та рекомендації щодо виправлення ситуації

Звіт про дефект (і сам дефект разом з ним) проходить певні стадії життєвого циклу:

- Виявлено (submitted) - початковий стан звіту (іноді називається «Новий» (new)), в якому він знаходиться відразу після створення. Деякі засоби також дозволяють спочатку створювати чернетку (draft) і лише потім публікувати звіт.
- Призначено (assigned) - в цей стан звіт переходить з моменту, коли хтось з проектної команди призначається відповідальним за виправлення дефекту. Призначення відповідального проводиться або рішенням лідера команди розробки, або колегіально, або за добровільним принципом, або іншим прийнятим в команді способом або виконується автоматично на основі певних правил.
- Виправлений (fixed) - в цей стан звіт переводить відповідальний за виправлення дефекту член команди після виконання відповідних дій з виправлення.
- Перевірено (verified) - в цей стан звіт переводить тестувальник, засвідчуючи, що дефект насправді був усунутий. Як правило, таку перевірку виконує тестувальник, що спочатку написав звіт про дефект.

З приводу того, чи повинен перевіряти факт усунення дефекту саме той тестувальник, який його виявив, або обов'язково інший, є багато «священних воєн». Прихильники другого варіанта стверджують, що свіжий погляд людини, раніше не знайомої з даним дефектом, дозволяє її в процесі верифікації з великою ймовірністю виявити нові дефекти.

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

Закрито (closed) - стан звіту, що означає, що з даного дефекту не планується ніяких подальших дій. Тут є деякі розбіжності в життєвому циклі, прийнятому в різних інструментальних засобах управління звітами про дефекти:

- У деяких засобах існують обидва стану - «Перевірено» і «Закрито», щоб підкреслити, що в стані «Перевірено» ще можуть знадобитися якісь додаткові дії (обговорення, додаткові перевірки в нових білдах і т.д.), в той час як стан «Закрито» означає «з дефектом покінчено, більше до цього питання не повертаємося».
- У деяких засобах одного з станів немає.
- У деяких засобах в стан «Закрито» або «відхилений» звіт про дефекті може бути переведений з безлічі попередніх станів з резолюціями на зразок:
  - «Не є дефектом» - додаток так і повинно працювати, описана поведінка не є аномальним.
  - «Дублікат» - даний дефект вже описаний 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 області повноважень команди розробників, наприклад існує проблема в операційній системі або апаратне забезпечення, вплив якої усунути розумними способами неможливо. Як було тільки що підкреслено, в деяких засобах звіт про дефект в подібних випадках буде переведений в стан «Закрито», в деяких - в стан «відхилений», в деяких - частина випадків закріплена за станом «Закрито», частина - за «відхилений».

Відкрито заново (reopened) - в цей стан (як правило, зі стану «Виправлено») звіт переводить тестувальник, засвідчуючи, що дефект як і раніше відтворюється на білді, в якому він вже повинен бути виправлений.

- Рекомендований до відхилення (to be declined) - в цей стан звіт про дефект може бути переведений з безлічі інших станів з метою винести на розгляд питання про відхилення звіту по тій або іншій причині. Якщо рекомендація є обгрунтованою, звіт переводиться в стан «Відхилений» (див. Наступний пункт).
- Відхилений (declined) - в цей стан звіт переводиться в випадках, докладно описаних в пункті «Закрито», якщо засіб управління звітами про дефекти передбачає використання цього стану замість стану «Закрито» для тих чи інших резолюцій по звіту.

Відкладений (deferred) - в цей стан звіт переводиться в разі, якщо виправлення дефекту в найближчий час є нераціональним або НЕ є можливим, однак є підстави вважати, що в доступному для огляду майбутньому ситуація виправиться (вийде нова версія бібліотеки, повернеться з відпустки фахівець з якоїсь технології, зміняться вимоги замовника і т.д.)

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.

- ідентифікатор
- короткий опис
- докладний опис
- кроки по відтворенню
- відтворюваність
- важливість
- терміновість
- симптом
- можливість обійти
- коментар
- додатки

Ідентифікатор (identifier) являє собою унікальне значення, що дозволяє однозначно відрізнити один звіт про дефект від іншого і використовується під всілякими посиланнями. У загальному випадку ідентифікатор звіту про дефект може являти собою просто унікальний номер, але (якщо дозволяє інструментальний засіб управління звітами) може бути і куди складнішим: включати префікси, суфікси і інші осмислені компоненти, що дозволяють швидко визначити суть дефекту і частину програми (або вимог), до якої він належить.

Короткий опис (резюме) повиннен бути в маскимально лаконічній формі дати відповідь на питання «Що відбулось?» «Де відбулось»? «За яких умов? ».

Наприклад: «Відсутній логотип на сторінці Привітання, якщо користувач є адміністратором »:

- Що відбулось? Відсутній логотип.
- Де це відбулось? На сторінці Привітання.
- За яких це відбулось? Якщо користувач є адміністратором.

Однією з найбільших проблем для початківців тестувальників єсаме заповнення поля «короткий опис», яке одночасно має:

- 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 запитання, які застосовні 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 був виявлений дефект;
- по можливості не дублювати інформацію інших дефектів (і навіть не бути схожими на них), щоб дефекти було складно переплутати або порахувати дублікатами один одного;
- бути закінченим реченням, побудованим за відповідними правилами граматики.

Для створення хороших коротких описів дефектів рекомендується користуватися таким алгоритмом:

1. Повноцінно зрозуміти суть проблеми. До тих пір, поки у тестувальника немає чіткого, кристально чистого розуміння того, «що зламалося», писати звіт про дефекті взагалі навряд чи варто.

2. Сформулювати докладний опис (description) дефекту - спочатку без оглядки на довжину отриманого тексту.

3. Прибрати з отриманого докладного опису все зайве, уточнити важливі деталі.

4. Виділити в докладному описі слова (словосполучення, фрагменти фраз), що відповідають на питання, «що, де і за яких умов сталося».

5. Оформити отримане в 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 і скорочень).

До речі, речення написане англійською мовою майже завжди буде коротшим за речення написане українською

Ситуація 1

Тестуванню підлягає якийсь веб-додаток, поле опису товару повинне допускати введення максимум 250 символів; в процесі тестування виявилося, що цього обмеження немає.

1. Суть проблеми: дослідження показало, що ні на клієнтській, ні на серверній частині немає ніяких механізмів, які перевіряють і / або обмежують довжину введених в поле «Інформація» даних.

2. Початковий варіант докладного опису: в клієнтській і серверній частині додатку відсутні перевірка і обмеження довжини даних, що вводяться в поле «Інформація» на сторінці http: // testapplication / admin / goods / edit /.

3. Кінцевий варіант докладного опису:

- Фактичний результат: в описі товару (поле «Інформація», http: // testapplication / admin / goods / edit /) відсутні перевірка і обмеження довжини тексту, що вводиться (MAX = 250 символів).
- Очікуваний результат: у разі спроби введення 251+ символів виводиться повідомлення про помилку.

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

- Що: відсутні перевірка і обмеження довжини тексту, що вводиться.
- Де: опис товару, поле «Інформація», http: // testapplication / admin / goods / edit /.
- За яких умов: - (в даному випадку дефект присутній завжди, незалежно від яких би то не було особливих умов).

5. Первинна формулювання: відсутні перевірка і обмеження максимальної довжини тексту, що вводиться в поле «Інформація» опису товару.

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

Ситуація 2

Спроба відкрити в додатку порожній файл призводить до краху клієнтської частини програми і втрати незбережених призначених для користувача даних на сервері.

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

2. Початковий варіант докладного опису: некоректний аналіз відкритого клієнтом файлу призводить до краху клієнта і незворотної втрати поточної сесії з сервером.

3. Кінцевий варіант докладного опису:

- Фактичний результат: відсутність перевірки коректності відкритого клієнтською частиною програми файлу (в тому числі порожнього) призводить до краху клієнтської частини і незворотної втрати поточної сесії з сервером (див. BR852345).
- Очікуваний результат: проводиться аналіз структури відкритого файлу; в разі виявлення проблем відображається повідомлення про неможливість відкриття файлу.

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

- Що: крах клієнтської частини програми.
- Де: - (конкретне місце в додатку визначити навряд чи можливо).
- За яких умов: при відкритті порожнього або пошкодженого файлу.

5. Первинна формулювання: відсутність перевірки коректності відкритого файлу призводить до краху клієнтської частини програми і втрати призначених для користувача даних.

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

Ситуація 3

Вкрай рідко з абсолютно незрозумілих причин на сайті порушується відображення всього українського тексту (як статичних написів, так і даних з бази даних, що генеруються динамічно і т.д. - все «стає питаннячко»).

1. Суть проблеми: фреймворк, на якому побудований сайт, підвантажує специфічні шрифти з віддаленого сервера; якщо з'єднання обривається, потрібні шрифт не завантажуються, і використовуються типові шрифти, в яких немає кириличних символів.

2. Початковий варіант докладного опису: помилка завантаження шрифтів з віддаленого сервера призводить до використання локальних несумісних з необхідною 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 завантажуються завжди (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 завантажуються шрифти.

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

6. Скорочення (підсумковий короткий опис): невірний показ українського тексту при недоступності зовнішніх шрифтів. Англійський варіант: wrong presentation of Ukrainian text in case of external fonts inaccessibility.

Детальний опис (description) представляє в розгорнутому вигляді необхідну інформацію про дефект, а також (обов'язково!) Опис фактичного результату, очікуваного результату і посилання на вимогу (якщо це можливо).

Приклад детального опису:

Якщо в систему входить адміністратор, на сторінці привітання відсутній логотип.

Фактичний результат: логотип відсутній в лівому верхньому кутку сторінки.

Очікуваний результат: логотип відображається в лівому верхньому кутку сторінки.

Вимога: R245.3.23b.

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

Кроки по відтворенню (steps to reproduce, STR) описують дії, які необхідно виконати для відтворення дефекту. Це поле схоже на кроки тест-кейса, за винятком однієї важливого відмінності: тут дії прописуються максимально докладно, із заначенням конкретних вхідних значень і найдрібніших деталей, тому що відсутність цієї інформації в складних випадках може привести до неможливості відтворення дефекту.

Приклад кроків відтворення:

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

2. Авторизуватися з ім'ям «defaultadmin» і паролем «dapassword».

Дефект: в лівому верхньому кутку сторінки відсутній логотип (замість нього відображається порожній простір з написом «logo»).

Відтворюваність (reproducibility) показує, чи при кожному проходженні по кроках відтворення дефекту вдається викликати його прояв. Це поле приймає всього два значення: завжди (always) або іноді (sometimes).

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

- тестувальнику потрібно витратити багато часу на те, щоб упевнитися в наявності дефекту (тому що одноразовий збій в роботі програми міг бути викликаний величезною кількістю сторонніх причин).
- Розробникові теж потрібно витратити час, щоб домогтися прояву дефекту і переконатися в його наявності. Після внесення виправлень у додаток розробник фактично повинен покладатися тільки на свій професіоналізм, тому що навіть багаторазове проходження по кроках відтворення в такому випадку не гарантує, що дефект був виправлений (можливо, через ще 10-20 повторень він би проявився).
- тестувальнику, що верифікує виправлення дефекту і зовсім залишається вірити розробнику на слово з тієї ж самої причини: навіть якщо він спробує відтворити дефект 100 раз і потім припинить спроби, може так статися, що на 101-й раз дефект все ж відтворився б.

Важливість (severity) показує ступінь шкоди, яка завдається проекту існуванням дефекту.

У загальному випадку виділяють наступні градації важливості:

- Критична (critical) - існування дефекту призводить до масштабних наслідків катастрофічного характеру, наприклад: втрата даних, розкриття конфіденційної інформації, порушення ключовий функціональності програми і т.д.
- Висока (major) - існування дефекту приносить відчутні незручності багатьом користувачам в рамках їх типової діяльності, наприклад: недоступність вставки з буфера обміну, непрацездатність загальноприйнятих клавіатурних комбінацій, необхідність перезапуску програми при виконанні типових сценаріїв роботи.
- Середня (medium) - існування дефекту слабо впливає на типові сценарії роботи користувачів, і / або існує обхідний шлях досягнення мети, наприклад: діалогове вікно не закривається автоматично після натискання кнопок «OK» / «Cancel», при роздруківці декількох документів поспіль НЕ зберігається значення поля «Двосторонній друк», переплутані напрямки угруповань по якомусь полю таблиці.
- Низька (minor) - існування дефекту рідко виявляється незначним відсотком користувачів і (майже) не впливає на їх роботу, наприклад: 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 (priority) показує, як швидко дефект повинен бути усунений.

У загальному випадку виділяють наступні градації терміновості:

- Найвища (ASAP, as soon as possible) терміновість вказує на необхідність усунути дефект настільки швидко, наскільки це можливо. Залежно від контексту «настільки швидко, наскільки можливо» може варіюватися від «в найближчому білді» до одиниць хвилин.
- Висока (high) терміновість означає, що дефект слід виправити позачергово, тому що його існування або вже об'єктивно заважає роботі, або почне створювати такі перешкоди в самому найближчому майбутньому.
- Звичайна (normal) терміновість означає, що дефект слід виправити в порядку загальної черговості. Таке значення терміновості отримує більшість дефектів.
- Низька (low) терміновість означає, що в доступному для огляду майбутньому виправлення даного дефекту не зробить істотного впливу на підвищення якості продукту

Симптом (symptom) - дозволяє класифікувати дефекти по їх типовому прояву. Не існує ніякого загальноприйнятого списку симптомів. Більш того, далеко не в кожному інструментальному засобі управління звітами про дефектах є таке поле, а там, де воно є, його можна налаштувати.

Часто зустрічається питання про те, чи може у одного дефекту бути відразу кілька симптомів. Так може. Наприклад, крах системи дуже часто веде до втрати або пошкодження даних. Але в більшості інструментальних засобів управління звітами про дефекти значення поля «Симптом» вибирається зі списку, і тому немає можливості вказати два і більше симптоми одного дефекту.

У такій ситуації рекомендується вибирати або симптом, який найкраще описує суть ситуації, або «найбільш небезпечний» симптом (наприклад, недружню поведінку, що складається в тому, що програма не потребує підтвердження перезапису існуючого файлу, призводить до втрати даних; тут «втрата даних» куди доречніше, ніж «недружня поведінка»).

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

Коментар (comments, additional info) - може містити будь-які корисні для розуміння і виправлення дефекту дані. Іншими словами, сюди можна писати все те, що не можна писати в інші поля.

Вкладення (attachments) - являє собою не стільки поле, скільки список прикріплених до звіту про дефект додатків (копій екрану, що викликають збій файлів і т.д.)

Загальні рекомендації по формуванню програм такі:

- Якщо ви вагаєтесь, робити або не робити вкладення, краще зробіть.
- Обов'язково додавайте т.зв. «Проблемні артефакти» (наприклад, файли, які додаток обробляє некоректно).

- Якщо ви докладаєте копію екрану:
  - Найчастіше вам буде потрібна копія активного вікна (Alt + PrintScreen), а не тільки екрану (PrintScreen).
  - Обріжте все зайве (використовуйте Snipping Tool або Paint в Windows, Pinta або XPaint в Linux).
  - Відзначте на копії екрану проблемні місця (обведіть, намалюйте стрілку, додайте напис - зробіть все необхідне, щоб з першого погляду проблема була помітна і зрозуміла).
  - У деяких випадках варто зробити одне велике зображення з декількох копій екрану (розмістивши їх послідовно), щоб показати процес відтворення дефекту. Альтернативою цього рішення є створення декількох копій екрану, 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 в форматі JPG (якщо важлива економія обсягу даних) або PNG (якщо важлива точна передача картинки без спотворень).

Якщо ви докладаєте 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 налаштування кодеків так, щоб отримати мінімальний розмір ролика при збереженні достатньої якості зображення.

- Експериментуйте з різними інструментами створення копій екрану і записів відеороликів з тим, що відбувається на екрані. Виберіть найбільш зручний для вас програмне забезпечення і привчіть себе постійно його використовувати.

Властивості якісних звітів про дефекти

Звіт про дефект може виявитися неякісним (а отже, ймовірність виправлення дефекту знизиться), якщо в ньому порушено одна з наступних властивостей.

- Ретельне заповнення всіх полів точної і коректної інформацією.
- Правильна технічна мова.
- Специфічність опису кроків.
- Відсутність зайвих дій і / або їх довгих описів.
- Відсутність дублікатів.
- Очевидність і зрозумілість.
- Відстежуваність.
- Окремі звіти для кожного нового дефекту.
- Відповідність прийнятим шаблонами оформлення і традиціям

Логіка створення ефективних звітів про дефекти

При створенні звіту про дефект рекомендується слідувати наступним алгоритмом:

0. Виявити дефект .

1. Зрозуміти суть проблеми.

2. Відтворити дефект.

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

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

5. Заповнити поля звіту, починаючи з докладного опису.

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

7. Ще раз перечитати звіт, тому що в пункті 6 ви точно щось упустили .

Зрозуміти суть проблеми

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

Відтворити дефект

Ця дія не тільки допоможе в подальшому правильно заповнити поле «Відтворюваність», а й дозволить уникнути неприємної ситуації, в якій за дефект додатку буде прийнятий якийсь короткочасний збій, який (швидше за все) стався десь у вашому комп'ютері або в іншій частині IT-інфраструктури, яка не має стосунку до тестованого додатку.

Перевірити наявність опису знайденого вами дефекту

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

Сформулювати суть проблеми

Формулювання проблеми у вигляді «що зробили (кроки по відтворенню), що отримали (фактичний результат в докладному описі), що очікували отримати (очікуваний результат в докладному описі) »дозволяє не тільки підготувати дані для заповнення полів звіту, а й ще краще зрозуміти суть проблеми.

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

- Прозорість і зрозумілість: слідуючи цій формулі, ви готуєте саме дані для звіту про дефект, що не скочуючись у розлогі абстрактні міркування.
- Легкість верифікації дефекту: розробник, використовуючи ці дані, може швидко відтворити дефект, а тестувальник після виправлення дефекту упевнитися, що той дійсно виправлений.
- Очевидність для розробників: ще до спроби відтворення дефекту видно, чи дійсно описане є дефектом, або тестувальник десь помилився, записавши в дефекти коректну поведінку програми.
- Позбавлення від зайвої безглуздою комунікації: докладні відповіді на «Що зробили, що отримали, що очікували отримати» дозволяють вирішувати проблему і усувати дефект без необхідності запиту, пошуку та обговорення додаткових відомостей.
- Простота: на фінальних стадіях тестування із залученням кінцевих користувачів можна відчутно підвищити ефективність надходить зворотного зв'язку, 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 з написання звіту.

Заповнити поля звіту

Починати краще за все з детального опису, тому що в процесі заповнення цього поля може виявитися безліч додаткових деталей, а також з'являться думки з приводу формулювання стисненого та інформативного короткого опису.

Якщо ви розумієте, що для заповнення якогось поля у вас не вистачає даних, проведіть додаткове дослідження. Якщо і воно не допомогло, опишіть у відповідному полі (якщо воно текстове), чому ви не впевнені у його заповненням, або (якщо поле є список) виберіть значення, яке, на ваш погляд, найкраще характеризує проблему (в деяких випадках інструментальний засіб дозволяє вибрати значення на зразок «невідомо», тоді виберіть його).

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

Перечитати звіт (і ще раз перечитати звіт)

Після того як все написано, заповнене і підготовлене, ще раз уважно перечитайте написане. Дуже часто ви зможете виявити, що в процесі доопрацювання тексту десь вийшли логічні нестиковки або дублювання, десь вам захочеться поліпшити формулювання, десь щось поміняти.

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

Practice

URL: http://prestashop.qatestlab.com.ua/uk/

1. Find 10 bugs;
2. Write bug-reports for all of them;

Practice 13.03

URL: http://prestashop.qatestlab.com.ua/uk/

1. Go to https://41pi2019.atlassian.net
2. Start your own project using Scrum (name must include your lastname);
3. Create 3-4 epics;
4. Add 10-12 user stories to each epic;
5. Break user stories to technical tasks inside one epic;
6. Find 10 bugs in prestashop and pull its into your Jira project;