цілі тестування

Підвищити ймовірність того, що додаток, призначене для тестування, буде працювати правильно при будь-яких обставинах.

Підвищити ймовірність того, що додаток, призначене для тестування, буде відповідати всім описаним вимогам.

Надання актуальної інформації про стан продукту на даний момент.

Етапи тестування:

1. Аналіз продукту

2. Робота з вимогами

3. Розробка стратегії тестування

і планування процедур контролю якості

4. Створення тестової документації

5. Тестування прототипу

6. Основне тестування

7. Стабілізація

8. Експлуатація

Тест план (Test Plan) - це документ, що описує весь обсяг робіт з тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи обладнання, спеціальних знань, а також оцінки ризиків з варіантами їх вирішення .

Відповідає на питання:

Що треба тестувати?

Що будете тестувати?

Як будете тестувати?

Коли будете тестувати?

Критерії початку тестування.

Критерії закінчення тестування.

Основні пункти тест плану

У стандарті IEEE 829 перераховані пункти, з яких повинен (нехай - може) складатися тест-план:

a) Test plan identifier;

b) Introduction;

c) Test items;

d) Features to be tested;

e) Features not to be tested;

f) Approach;

g) Item pass / fail criteria;

h) Suspension criteria and resumption requirements;

i) Test deliverables;

j) Testing tasks;

k) Environmental needs;

l) Responsibilities;

m) Staffing and training needs;

n) Schedule;

o) Risks and contingencies;

p) Approvals.

Тест дизайн - це етап процесу тестування ПО, на якому проектуються і створюються тестові сценарії (тест кейси), відповідно до визначених раніше критеріями якості та цілями тестування.

Ролі, відповідальні за тест дизайн:

• Тест аналітик - визначає «ЩО тестувати?»

• Тест дизайнер - визначає «ЯК тестувати?»

Техніки тест дизайну

• Еквівалентна Поділ (Equivalence Partitioning - EP). Як приклад, у вас є діапазон допустимих значень від 1 до 10, ви повинні вибрати один вірний значення всередині інтервалу, скажімо, 5, і один невірний значення поза інтервалу - 0.

• Аналіз Граничних Значний (Boundary Value Analysis - BVA). Якщо взяти приклад вище, в якості значень для позитивного тестування виберемо мінімальну і максимальну межі (1 і 10), і значення більше і менше кордонів (0 і 11). Аналіз Граничний значень може бути застосований до полів, записів, файлів, або до будь-якого роду сутностей мають обмеження.

• Причина / Слідство (Cause / Effect - CE). Це, як правило, введення комбінацій умов (причин), для отримання відповіді від системи (Слідство). Наприклад, ви перевіряєте можливість додавати клієнта, використовуючи певну екранну форму. Для цього вам необхідно буде ввести кілька полів, таких як «Ім'я», «Адреса», «Номер Телефону» а потім, натиснути кнопку «Додати» - це «Причина». Після натискання кнопки «Додати», система додає клієнта в базу даних і показує його номер на екрані - це «Слідство».

• Передбачення помилки (Error Guessing - EG). Це коли тестувальник використовує свої знання системи і здатність до інтерпретації специфікації на предмет того, щоб «вгадати» за яких вхідних умовах система може видати помилку. Наприклад, специфікація каже: «користувач повинен ввести код». Тестувальник буде думати: «Що, якщо я не введу код?», «Що, якщо я введу неправильний код? ", і так далі. Це і є передбачення помилки.

• Повне тестування (Exhaustive Testing - ET) - це крайній випадок. В межах цієї техніки ви повинні перевірити всі можливі комбінації вхідних значень, і в принципі, це повинно знайти всі проблеми. На практиці застосування цього методу не представляється можливим, через величезної кількості вхідних значень.

• Попарне тестування (Pairwise Testing) - це техніка формування наборів тестових даних. Сформулювати суть можна, наприклад, ось так: формування таких наборів даних, в яких кожне тестоване значення кожного з перевірених параметрів хоча б один раз поєднується з кожним тестованим значенням всіх інших перевіряються параметрів.

Баг Репорт (Bug Report) - це документ, що описує ситуацію або послідовність дій призвела до некоректної роботи об'єкта тестування, із зазначенням причин і очікуваного результату.

шапка

Короткий опис (Summary) короткий опис проблеми, явно вказує на причину і тип помилкової ситуації.

Проект (Project) Назва тестованого проекту

Компонент додатка (Component) Назва частини або функції тестованого продукту

Номер версії (Version) Версія на якій була знайдена помилка

Серйозність (Severity) Найбільш поширена п'ятирівнева система градації серйозності дефекту:

• S1 Блокуючий (Blocker)

• S2 Критичний (Critical)

• S3 Значний (Major)

• S4 Незначний (Minor)

• S5 Тривіальний (Trivial)

Пріоритет (Priority) Пріоритет дефекту:

• P1 Високий (High)

• P2 Середній (Medium)

• P3 Низький (Low)

Статус (Status) Статус бага. Залежить від процедури і життєвого циклу бага (bug workflow and life cycle)

Автор (Author) Творець баг репорт

Призначено на (Assigned To) Ім'я співробітника, призначеного на вирішення проблеми

оточення

ОС / Сервіс Пак і т.д. / Браузера + версія / ... Інформація про оточення, на якому був знайдений баг: операційна система, сервіс пак, для WEB тестування - ім'я і версія браузера і т.д.

...

опис

Кроки відтворення (Steps to Reproduce) Кроки, за якими можна легко відтворити ситуацію, яка призвела до помилки.

Фактичний Результат (Result) Результат, отриманий після проходження кроків до відтворення

Очікуваний результат (Expected Result) Очікуваний правильний результат

додатки

Прикріплений файл (Attachment) Файл з балками, скріншот або будь-який інший документ, який може допомогти прояснити причину помилки або вказати на спосіб вирішення проблеми

Severity vs Priority

Серйозність (Severity) - це атрибут, що характеризує вплив дефекту на працездатність програми.

Пріоритет (Priority) - це атрибут, який вказує на черговість виконання завдання або усунення дефекту. Можна сказати, що це інструмент менеджера з планування робіт. Чим вище пріоритет, тим швидше потрібно виправити дефект.

Severity виставляється тестувальником

Priority - менеджером, тімліда або замовником

Градація Серйозності дефекту (Severity)

S1 Блокуюча (Blocker)

Блокуюча помилка, що приводить додаток в неробочий стан, в результаті якого подальша робота з тестованої системою або її ключовими функціями стає неможлива. Рішення проблеми необхідно для подальшого функціонування системи.

S2 Критична (Critical)

Критична помилка, неправильно працює ключова бізнес логіка, діра в системі безпеки, проблема, яка призвела до тимчасового падіння сервера або приводить в неробочий стан деяку частину системи, без можливості вирішення проблеми, використовуючи інші вхідні точки. Рішення проблеми необхідно для подальшої роботи з ключовими функціями тестируемой системою.

S3 Значна (Major)

Значна помилка, частина основної бізнес логіки не функціонує належним чином. Помилка не критична або є можливість для роботи з тестованої функцією, використовуючи інші вхідні точки.

S4 Незначна (Minor)

Незначна помилка, що не порушує бізнес логіку тестируемой частини програми, очевидна проблема призначеного для користувача інтерфейсу.

S5 Тривіальна (Trivial)

Тривіальна помилка, яка не стосується бізнес логіки додатка, погано відтворена проблема, малопомітна посредствам призначеного для користувача інтерфейсу, проблема сторонніх бібліотек або сервісів, проблема, не надає ніякого впливу на загальну якість продукту.

Градація Пріоритету дефекту (Priority)

P1 Високий (High)

Помилка повинна бути виправлена ​​якомога швидше, тому що її наявність є критичною для проекту.

P2 Середній (Medium)

Помилка повинна бути виправлена, її наявність не є критичною, але вимагає обов'язкового рішення.

P3 Низький (Low)

Помилка повинна бути виправлена, її наявність не є критичною, і не потребує термінового вирішення.

рівні Тестування

1. Модульне тестування (Unit Testing)

Компонентний (модульне) тестування перевіряє функціональність і шукає дефекти в частинах додатка, які доступні і можуть бути протестовані по-окремо (модулі програм, об'єкти, класи, функції і т.д.).

2. Інтеграційний тестування (Integration Testing)

Перевіряється взаємодія між компонентами системи після проведення компонентного тестування.

3. Системне тестування (System Testing)

Основним завданням системного тестування є перевірка як функціональних, так і не функціональних вимог в системі в цілому. При цьому виявляються дефекти, такі як неправильне використання ресурсів системи, непередбачені комбінації даних користувача рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або неправильна функціональність, незручність використання і т.д.

4. Операційне тестування (Release Testing).

Навіть якщо система задовольняє всім вимогам, важливо переконатися в тому, що вона задовольняє потребам користувача і виконує свою роль в середовищі своєї експлуатації, як це було визначено в бізнес моделе системи. Слід врахувати, що і бізнес модель може містити помилки. Тому так важливо провести операційне тестування як фінальний крок валідації. Крім цього, тестування в середовищі експлуатації дозволяє виявити і нефункціональні проблеми, такі як: конфлікт з іншими системами, суміжними в області бізнесу або в програмних і електронних середовищах; недостатня продуктивність системи в середовищі експлуатації та ін. Очевидно, що знаходження подібних речей на стадії впровадження - критична і дорога проблема. Тому так важливо проведення не тільки верифікації, а й валідації, з самих ранніх етапів розробки ПЗ.

5. Приймальне тестування (Acceptance Testing)

Формальний процес тестування, який перевіряє відповідність системи вимогам і проводиться з метою:

• визначення чи задовольняє система приймальним критеріям;

• винесення рішення замовником або іншою уповноваженою особою приймається додаток чи ні.

Види / типи тестування

Функціональні види тестування

• Функціональне тестування (Functional testing)

• Тестування для користувача інтерфейсу (GUI Testing)

• Тестування безпеки (Security and Access Control Testing)

• Тестування взаємодії (Interoperability Testing)

Нефункціональні види тестування

• Всі види тестування продуктивності:

o тестування навантаження (Performance and Load Testing)

o стресове тестування (Stress Testing)

o тестування стабільності або надійності (Stability / Reliability Testing)

o об'ємне тестування (Volume Testing)

• Тестування установки (Installation testing)

• Тестування зручності користування (Usability Testing)

• Тестування на відмову і відновлення (Failover and Recovery Testing)

• Конфігураційне тестування (Configuration Testing)

Пов'язані зі змінами види тестування

• Димові тестування (Smoke Testing)

• Регресійне тестування (Regression Testing)

• Повторне тестування (Re-testing)

• Тестування збірки (Build Verification Test)

• Санітарне тестування або перевірка узгодженості / справності (Sanity Testing)

Функціональне тестування розглядає заздалегідь вказане поведінку і грунтується на аналізі специфікацій функціональності компонента або системи в цілому.

Тестування для користувача інтерфейсу (GUI Testing) - функціональна перевірка інтерфейсу на відповідність вимогам - розмір, шрифт, колір, consistent behavior.

Тестування безпеки - це стратегія тестування, яка використовується для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатки, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Тестування взаємодії (Interoperability Testing) - це функціональне тестування, що перевіряє здатність додатки взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування

Тестування навантаження - це автоматизоване тестування, що імітує роботу певної кількості бізнес користувачів на якомусь загальному (разделяемом ними) ресурсі.

Стресове тестування (Stress Testing) дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом в даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійне зміна конфігурації сервера. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності.

Об'ємне тестування (Volume Testing). Завданням об'ємного тестування є отримання оцен ки продуктивності при збільшенні обсягів даних в базі даних програми

Тестування стабільності або надійності (Stability / Reliability Testing). Завданням тестування стабільності (надійності) є перевірка працездатності програми при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження.

Тестування установки направлено на перевірку успішної інсталяції та настройки, а також поновлення або видалення програмного забезпечення.

Тестування зручності користування - це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості і привабливості для користувачів продукту, що розробляється в контексті заданих умов. Сюди також входить:

User eXperience (UX) - відчуття, яке відчувається користувачем під час використання цифрового продукту, в той час як User interface - це інструмент, що дозволяє здійснювати интеракцию «користувач - веб-ресурс».

Тестування на відмову і відновлення (Failover and Recovery Testing) перевіряє тестований продукт з точки зору здатності протистояти і успішно відновлюватися після можливих збоїв, що виникли в зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, в разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.

Конфігураційне тестування (Configuration Testing) - спеціальний вид тестування, спрямований на перевірку роботи програмного забезпечення при різних конфігураціях системи (заявлених платформах, підтримуваних драйвери, при різних конфігураціях комп'ютерів і т.д.)

Димові (Smoke) тестування розглядається як короткий цикл тестів, що виконується для підтвердження того, що після складання коду (нового або відредагованого) встановлюється додаток, стартує і виконує основні функції.

Регресійне тестування - це вид тестування спрямований на перевірку змін, зроблених в додатку або навколишньому середовищу (лагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатки), для підтвердження того факту, що існуюча раніше функціональність працює як і раніше. Регресійний можуть бути як функціональні, так і нефункціональні тести.

Повторне тестування - тестування, під час якого виконуються тестові сценарії, які виявили помилки під час останнього запуску, для підтвердження успішності виправлення цих помилок.

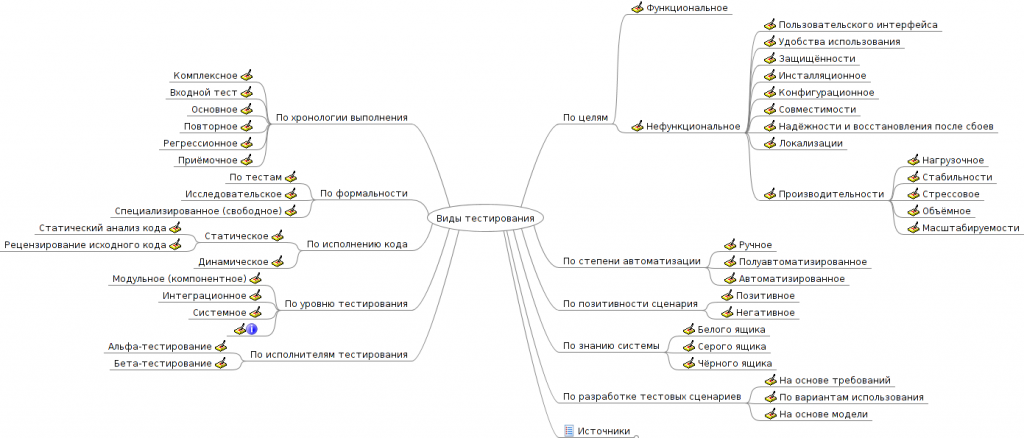
У чому різниця між regression testing і re-testing?

Re-testing - перевіряється виправлення багів

Regression testing - перевіряється те, що виправлення багів, а також будь-які зміни в коді програми, не вплинули на інші модулі ПО і не викликало нових багів.

Тестування збірки або Build Verification Test - тестування спрямоване на визначення відповідності, випущеної версії, критеріям якості для початку тестування. За своїми цілями є аналогом димових Тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглиб воно може проникати далі, в залежності від вимог до якості випущеної версії.

Санітарне тестування - це вузьконаправлене тестування достатню для доказу того, що конкретна функція працює згідно із заявленими в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або навколишньому середовищу. Зазвичай виконується вручну.



принципи тестування

Принцип 1 - Тестування демонструє наявність дефектів (Testing shows presence of defects)

Тестування може показати, що дефекти присутні, але не може довести, що їх немає. Тестування знижує ймовірність наявності дефектів, що знаходяться в програмному забезпеченні, але, навіть якщо дефекти не були виявлені, це не доводить його коректності.

Принцип 2 - Повне тестування недосяжно (Exhaustive testing is impossible)

Повне тестування з використанням всіх комбінацій вводів і передумов фізично неможливо, за винятком тривіальних випадків. Замість вичерпного тестування повинні використовуватися аналіз ризиків і розстановка пріоритетів, щоб більш точно сфокусувати зусилля по тестуванню.

Принцип 3 - Раннє тестування (Early testing)

Щоб знайти дефекти якомога раніше, активності по тестуванню повинні бути розпочаті якомога раніше в життєвому циклі розробки програмного забезпечення або системи, і повинні бути сфокусовані на певних цілях.

Принцип 4 - Скупчення дефектів (Defects clustering)

Зусилля тестування повинні бути зосереджені пропорційно очікуваній, а пізніше реальної щільності дефектів по модулях. Як правило, більша частина дефектів, виявлених при тестуванні або спричинили за собою основну кількість збоїв системи, міститься в невеликій кількості модулів.

Принцип 5 - Парадокс пестициду (Pesticide paradox)

Якщо одні й ті ж тести будуть проганяти багато разів, в кінцевому рахунку цей набір тестових сценаріїв більше не буде знаходити нових дефектів. Щоб подолати цей «парадокс пестициду», тестові сценарії повинні регулярно рецензуватися і коригуватися, нові тести повинні бути різнобічними, щоб охопити всі компоненти програмного забезпечення,

або системи, і знайти якомога більше дефектів.

Принцип 6 - Тестування залежить від контексту (Testing is concept depending)

Тестування виконується по-різному в залежності від контексту. Наприклад, програмне забезпечення, в якому критично важлива безпека, тестується інакше, ніж сайт електронної комерції.

Принцип 7 - Помилка про відсутність помилок (Absence-of-errors fallacy)

Виявлення і виправлення дефектів не допоможуть, якщо створена система не підходить користувачеві і не задовольняє його очікуванням і потребам.

Cтатіческое і динамічне тестування

Статична тестування відрізняється від динамічного тим, що виробляється без запуску програмного коду продукту. Тестування здійснюється шляхом аналізу програмного коду (code review) або скомпільованої коду. Аналіз може проводитися як вручну, так і за допомогою спеціальних інструментальних засобів. Метою аналізу є раннє виявлення помилок і потенційних проблем в продукті. Також до статичного тестування відноситься тестування специфікації та іншої документації.

Дослідницьке / ad-hoc тестування

Найпростіше визначення дослідницького тестування - це розробка і виконання тестів в один і той же час. Що є протилежністю сценарного підходу (з його зумовленими процедурами тестування, неважливо ручними або автоматизованими). Дослідницькі тести, на відміну від сценарних тестів, не визначені заздалегідь і не виконуються в точній відповідності з планом.

Різниця між ad hoc і exploratory testing в тому, що теоретично, ad hoc може провести будь-хто, а для проведення exploratory необхідно майстерність і володіння певними техніками. Зверніть увагу, що певні техніки це не тільки техніки тестування.

Вимоги - це специфікація (опис) того, що повинно бути реалізовано.

Вимоги описують те, що необхідно реалізувати, без деталізації технічної сторони рішення. Що, а не як.

Вимоги до вимог:

• Коректність

• недвозначно

• Повнота набору вимог

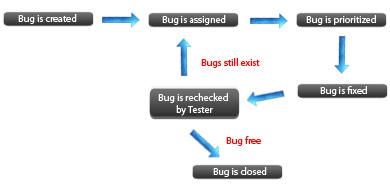
• Несуперечність набору вимог

• Верифікованість (тестопригодності)

• трассируемого

• зрозумілі

**Жизненный цикл бага**



Стадії розробки ПО - це етапи, які проходять команди розробників ПЗ, перш ніж програма стане доступною для широко кола користувачів. Розробка ПО починається з початкового етапу розробки (стадія «пре-альфа») і триває стадіями, на яких продукт допрацьовується і модернізується. Фінальним етапом цього процесу стає випуск на ринок остаточної версії програмного забезпечення ( «загальнодоступного релізу»).

Програмний продукт проходить наступні стадії:

• аналіз вимог до проекту;

• проектування;

• реалізація;

• тестування продукту;

• впровадження та підтримка.

Кожній стадії розробки ПО присвоюється певний порядковий номер. Також кожен етап має свою власну назву, яке характеризує готовність продукту на цій стадії.

Життєвий цикл розробки ПЗ:

• Пре-альфа

• Альфа

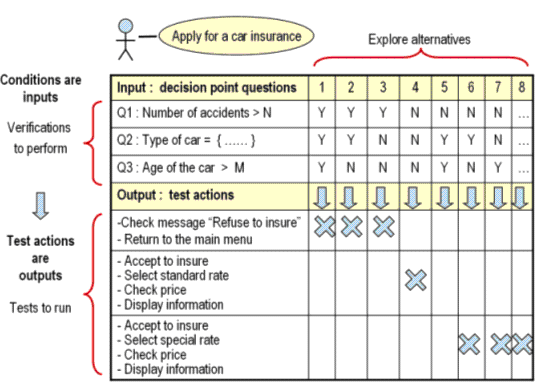
• Бета

• Реліз-кандидат

• Реліз

• Пост-реліз

Таблиця прийняття рішень (decision table) - чудовий інструмент для упорядкування складних бізнес вимог, які повинні бути реалізовані в продукті. У таблицях рішень представлений набір умов, одночасне виконання яких повинно привести до певної дії.\



ОБЗАЦ

Текст, як основне джерело інформації при роботі через клієнт

З якої сторони не поглянути, але особлива цінність мережі Інтернет полягає в тому, що вона є практично безмежним джерелом інформації. Частина цієї інформації представлена ​​у вигляді текстів, з якими, знову ж таки, користувач взаємодіє за допомогою клієнта.

Звичайно, значимість цього тестування не така велика в порівнянні з функціональним напрямом, але нехтувати ним не варто. На практиці мені довелося зустрітися з дуже серйозною помилкою: на продакшені одного з найбільших інтернет-гіпермаркетів Росії була допущена помилка, яка змінила горду назву величного міста Ярославль на сумне образливе слово. Назва служило частиною суб-меню, а тому було помітно всім жителям міста з півмільйонним населенням.

Так, ми не завжди маємо достатньо часу для вичитування всіх текстів, але в таких ситуаціях на допомогу приходять «SpellCheker-и» (програми для перевірки орфографії, онлайн або у вигляді плагінів для браузерів, або десктопні программи), наприклад, Яндекс.Спеллер чи Punto Switcher.