**Лабораторна робота №9-3**. Визначення рівнів тестування та переліку видів тестів на кожному рівні

**Мета:** Навчитися формувати формальні специфікації функціональних елементів

**Методичні рекомендації до виконання лабораторної роботи.**

1. Опрацювати теоретичні відомості.
2. Розглянути надані в теоретичних відомостях вили тестування та розподілити їх по рівнях тестування з урахуванням поділу на "підрівні", підходи та методи рівня. Заповнити таблицю:

|  |  |  |
| --- | --- | --- |
| №№ | Рівень тестування / підрівень / підхід / медод | Види тестування |
|  |  |  |
|  | … |  |
|  |  |  |

1. Робота повинна бути виконана згідно критеріїв оформлення документації та повинна містити

* Назва лабораторної роботи.
* Прізвище, група
* Назва проекту.
* Результати роботи оформлюються у вигляді таблиці:

По закінченню практичну роботу потрібно здати на перевірку викладачеві, надіславши електронною поштою на адресу [**t.i.lumpova@gmail.com**](mailto:t.i.lumpova@gmail.com) . Якщо викладач знаходить помилки чи неточності, він може повернути роботу на доопрацювання.

Файл з роботою повинен мати назву в такому форматі:

**QA<Номер групи><Номер лекції / практичної / лабораторної>[-<Номер завдання>][літера позначення типу роботи L – лекція, P –практична, R – лабораторна]<Прізвище англійською>**.. Наприклад, **QA4101Р**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт-"близнюків" відповідь буде зараховуватися першому за часом надсилання.

Тему в заголовку листа записати

**QA <Номер групи>-ЛР<Номер лабораторної>-<Прізвище >**

**Строк виконання цієї роботи ІПЗ-41 – 08.04.2024**

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Тестування на різних рівнях проводиться протягом усього життєвого циклу розробки і супроводу ПЗ. Рівень тестування визначає те, над чим виробляються тести: над окремим модулем, групою модулів або системою в цілому. Проведення тестування на всіх рівнях системи – це запорука успішної реалізації та здачі проекту.

**Рівні тестування (Testing levels):**

* Компонентне або Модульне тестування (Component testing or Unit testing)
* Інтеграційне тестування (Integration testing)
* Системне тестування (System testing)
* Приймальне тестування (Acceptance testing)

**Компонентне тестування** перевіряє функціональність і шукає дефекти в частинах програми, які доступні і можуть бути протестовані окремо (модулі програми, об’єкти, функції і т.д.)

Зазвичай компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити чи за підтримки середовищ розробки, таких як фреймовки (каркаси) для модульного тестування або інструменти для дебагу. Всі знайдені дефекти, як правило виправляються в коді без формального їх опису в системі багів (Bug Tracking System)

Один з найбільш ефективних підходів до компонентного (модульного) тестування – це підготовка автоматизованих тестів до початку основного кодування ПЗ. Це називається розробка від тестування (test-driven development) або підхід тестування спочатку (test first approach). При цьому підході створюються і інтегруються невеликі шматки коду, навпроти яких запускаються тести, написані до початку кодування. Розробка ведеться до тих пір поки всі тести не будуть успішними.

**Інтеграційне тестування** призначено для перевірки зв’язку між компонентами, а також взаємодії з різними частинами системи (операційної системи, обладнанням небудь зв’язку між різними системами).  
Рівні інтеграційного тестування:

* Компонентний інтеграційний рівень (Component Integration testing) перевіряє взаємодія між різними системами після проведення компонентного тестування.
* Системний інтеграційний рівень (System Integration testing) перевіряє взаємодію між різними системами після проведення системного тестування.

**Системне тестування** – основним завданням є перевірка як функціональних так і не функціональних вимог у системі в цілому. При цьому виявляються дефекти, такі як невірне використання ресурсів системи, непередбачені комбінації даних рівня користувача, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання тощо. Для мінімізації ризиків, пов’язаних з особливостями поведінки системи в тому чи іншому середовищі, під час тестування рекомендується використовувати оточення максимально наближене до того, на яке буде встановлений продукт після релізу. Існує два підходи до системного тестування:

* На базі вимог (requirements based) – Для кожної вимоги пишуться тестові випадки (test cases), перевіряючі виконання даної вимоги.
* На базі випадків використання (use case based) – На основі уявлення про способи використання продукту створюються випадки використання системи (Use Cases). По конкретному випадку використання можна визначити один або більше сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які повинні бути протестовані.

**Приймальне тестування** проводиться з метою: визначення чи задовольняє система приймальні критерії там винесення рішення замовником або іншою уповноваженою особою приймається програма чи ні.  
Приймальне тестування виконується відповідно до Плану приймальних Робіт.  
Рішення про проведення приймального тестування приймається, коли: продукт досяг необхідного рівня якості та замовник ознайомлений з Планом приймальних Робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов’язаних з проведенням приймального тестування, дата проведення, відповідальні і т.д.

**Методи приймального тестування:**

* Тестування замовником самостійно. Це ризиковано в тому плані що у замовника може не бути творчих ресурсів, а завантаження по поточним завданням може розтягти процес приймання.
* Тестування (Аудит) третьою стороною. Наймається спеціалізована компанія на тестуванні або підписується договір з конкурентом постачальника на надання послуг аудиту.
* Спільне тестування за сценаріями із замовником. Постачальник допомагає готувати пакет матеріалів для приймального тестування, готує команду замовника до методичного приймального тестування, контролює хід приймального тестування і терміни його виконання. Присутність інженера з тестування з боку виконавця допоможе краще зафіксувати розбіжності, зауваження та виявлені дефекти.

Фаза приймального тестування триває до тих пір, поки замовник не виносить рішення про відправлення програми на доопрацювання або реліз програми.

Незважаючи на те, що приймання знаходиться в кінці етапу (а в невеликих проектах і в кінці проекту) – готуватися до неї потрібно заздалегідь і перший прогін потрібно робити трохи раніше – щоб визначитися з повнотою і якістю робочого набору артефактів приймання, привчити до нього замовника, заздалегідь виявити можливі проблеми в приймальних тестах або в продукті.

**ВИДИ ТЕСТУВАННЯ**

Існує кілька ознак, за якими класифікують тестування програмного забезпечення на види тестування. Згідно книги "Тестування. Основні визначення. Методичний посібник КПІ" Дідковської (http://mmsa.kpi.ua/sites/default/files/disciplines/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0%20%D1%96%20%D1%82%D0%B5%D1%81%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%20%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC/didkovska\_m\_v\_testing\_definition\_part1.pdf).

### **За об’єктом тестування виділяють наступні види тестування ПЗ:**

* функціональне тестування **(functional testing);**

**Functional testing** (Функціональне тестування) — передбачає аналіз функціональних характеристик додатка та перевірку на невідповідності між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і бізнес-вимоги. Функціональне тестування імітує фактичне використання системи.

Функціональні тести можуть проводитися на усіх рівнях тестування (компонентному, інтеграційному, системному, приймальному). Як правило, ці функції описуються в вимогах, функціональних специфікаціях або у вигляді випадків використання системи (Use case).

* дослідницьке тестування **(exploratory testing);**

**Exploratory testing або дослідницьке тестування —** це одночасне вивчення програмного продукту, проектування тестів і їх виконання.

Дослідницьке тестування — це спосіб проектування тестів, при якому тестувальник активно контролює проектування тестів в той час, як ці тести виконуються, і використовує отриману під час тестування інформацію для проектування нових наступних і поліпшених тестів.

Якщо кожен наступний тест, який ми виконуємо, вибудовується за результатами попереднього тесту, це означає, що ми використовуємо дослідне тестування.

Зазначимо, слід мати на увазі що Exploratory testing саме по собі воно не є як таким видом чи методикою тестування в чистому вигляді. Це, скоріше, підхід або спосіб мислення. Що є протилежністю сценарного підходу (з його чітко визначеними процедурами тестування, неважливо ручними або автоматизованими). У той же час дослідницьке тестування гарантує, що вказані у системних вимогах функції працюватимуть належним чином.

***Тест*** — це те, що дає нову інформацію про програму. Один той самий тест виконаний двічі вже не можна назвати тестом, скоріше за все це вже буде перевірка. [Джеймс Бах]

* тестування продуктивності **(performance testing);**

**Performance Testing (Тестування продуктивності)** — це тип нефункціонального тестування, в якому продуктивність системи оцінюється при очікуваному або більш високому навантаженні в умовах різних сценаріїв поведінки. Ціль Performance Testing перевірити швидкодію реакції системи на певний рівень навантаження, працездатність, стабільність. Параметри оцінки продуктивності: час відгуку, надійність, використання ресурсу, масштабованість тощо.

Тестування продуктивності дозволяє виявити можливі уразливості та недоліки в програмі з метою запобігання їх впливу на роботу системи під час використання.

* навантажувальне тестування **(load testing);**

**Load Testing (Тестування навантаження)** — це тип Performance Testing й техніка автоматизованого тестування метою якого є перевірити, що наша програма працює у відповідності до вимог за нормальних умов, тобто при звичайному очікуваному користувацькому навантаженні.

Під час перевірки навантаження ми оцінюємо час відгуку на запити різних типів, пропускну спроможність, використання ресурсів, частоту помилок та інші параметри програми по бажанню. По результатах формуємо Метрики у якості звіту для зацікавлених сторін.

Відмінності між Load and Performance testing

Існують думки у мережі, що цього типу тестування виділяти не слід, бо бачите це «дубль» самого Performance Testing. Але існує дещо інша думка стосовно Load Testing.

Тестування продуктивності — це більш комплексне поняття. Коли ми говоримо про Performance Testing, то ми маємо на увазі перевірку системи за різних умов: нормальних показниках, максимальному навантаженні згідно документації та гітотечичних (стресових) умов **Тестування навантаження (Load Testing) виконується тільки для нормальних умов!!!**.

* димне/димове тестування **(smoke testing);**

**Димове тестування (Smoke testing)** також є аналогом тестування складання (Build Verification Testing) – це тип тестування програмного забезпечення, який включає невичерпний набір тестів, спрямованих на перевірку роботи найбільш важливих, критичних функцій в системі. Результат цього тестування використовується для визначення того, чи досить стабільна збірка, щоб продовжити подальше тестування.

Димові тести здійснюються в тих випадках, коли тестувальники отримують нову версію (білд) програми на тестування, при цьому вважаючи її відносно нестабільною. На цьому етапі необхідно переконатися, що надзвичайно важливі функції AUT (Application Under Test) працюють згідно з очікуваннями. Концепція цього виду тестування полягає в тому, щоб виявити серйозні проблеми якомога раніше і відхилити цю версію (білд) або повернути її на доопрацювання на ранньому етапі тестування. Це необхідно, щоб не поглиблюватися в складні тести і не витрачати час на явно неякісне програмне забезпечення.

Метою димового тестування є перевірка надійності і стабільності програмного забезпечення загалом, щоб дозволити подальше проведення ретельнішого тестування.

***Переваги димового тестування:***

* відразу знаходяться проблеми інтеграції;,
* тестування розкриває проблеми на ранній стадії;
* забезпечує певний рівень впевненості в тому, що зміни в програмному забезпеченні не зробили несприятливої дії на основні області системи;
* збій при димовому тестуванні призводить до негайної відмови від цієї збірки програмного забезпечення.
* стрес-тестування **(stress testing);**

**Stress Testing** — це також один із типів Performance Testing. Стрес-тестування передбачає тестування продуктивності, шляхом збільшення робочого навантаження на програму більше ніж очікується — створення штучного контрольованого стресу для неї. Стрес-тестування проводиться для виявлення витоків пам’яті та перевірки надійності програми.

* тестування стабільності **(stability / endurance / soak testing);**

**Soak Testing** — схожий тип тестування на попередній **Reliability Testing**, але різниця полягає в тому що досліджується працездатність на витривалість при навантаженні вище середнього значення. Soak Testing потрібне щоб дізнатися чи зможе система витримувати навантаження, наприклад високими об’ємами оброблюваних даних та побачити, що відбуватиметься поза дизайнерськими очікуваннями. **Reliability Testing** — це тип тестування програмного забезпечення на витривалість, який досліджує працездатність додатку при тривалій багатогодинній роботі, при середньому для програми навантаженні. Ціль виявити витоки ресурсів. Тобто у процесі тестування ретельно моніторяться ресурси системи (пам’ять, процесор, завантаження диску, файлові дескриптори, сокети та ін. показники).

**Endurance Testing, Stability Testing —** синоніми до **Soak Testing.**

* тестування зручності використання **(usability testing);**

**Usability Testing** — це тестування додатку на предмет: краси, корисності, зрозумілості, легкості та зручності використання користувачами. Наскільки інтерфейс додатку їм імпонує. Його мета є віднайти дефекти та покращити зручність використання (так зване "юзабіліті") додатку.

Найкраще, коли зручність використання тестують кінцеві користувачі, а не тестувальники. Завдання тестувальника полягає в підготовці набору практичних значень, розробці детальних користувацьких інструкцій пов’язаних з реальною діяльністю, повторюваних тестових завдань, які повинен буде виконати кожен користувач.

Важливу роль у Usability Testing відіграє збір та аналіз даних, зведення та інтерпретація результатів спостережень, своїх тестерських, зроблених користувачами, з відповідей користувачів на анкети після проведення тестів тощо. На основі цього аналізу розробники вносять в програмне забезпечення відповідні зміни.

* тестування інтерфейсу користувача **(ui testing);**

Інтерфейс користувача — це зовнішній вигляд програми. А як відомо зустрічають по «одежі». Отже, **UI Testing (Тестування інтерфейсу)** — це вид тестування, який має на меті знайти дефекти графічного інтерфейсу користувача та перевіряти, чи відповідає GUI специфікаціям.

**UI Testing (Тестування користувацького) інтерфейсу включає в себе:**

* перевірку того, як різні елементи інтерфейсу (панелі інструментів, меню, зображення, діалоги, поля введення, списки, кнопки тощо),
* реагують на дії користувача; перевірка того, яким чином додаток обробляє дії користувача з клавіатури та миші.
* тестування безпеки **(security testing);**

**Тестування Безпеки (Security testing)** — це вид тестування програмного забезпечення, який має на меті оцінити та перевірити цілісність системи від стороннього вторгнення. А саме автентифікацію, авторизацію, доступність, конфіденційність і безвідмовність аплікації під тестами.

* тестування локалізації **(localization testing);**

**Тестування локалізації** — це тестування за яким ми оцінюємо налаштування програми (локалізовану версію програми) на певній мові. Наприклад, чи правильно зроблений переклад?«Кракозябри» не з’являються? Стандарт мови правильно увімкнули? Для кирилиці UTF-8.

* тестування сумісності **(compatibility testing).**

**Compatibility Testing (Тестування сумісності)** — тестування програмного забезпечення, призначене щоб побачити, наскільки сумісне програмне забезпечення з певним середовищем — операційною системою, платформою чи обладнанням.

Наприклад:

* Мережевими пристроями;
* Периферія (принтери, мишка, USB, веб-камери);
* Операційними системами (Unix, Windows, MacOS, …);
* Бази даних (Oracle, MS SQL, MySQL, …);
* Системним програмним забезпеченням (веб-сервер, файрволл, антивірус, …);
* Браузерами (Internet Explorer, Firefox, Opera, Chrome, Safari).

### **У залежності від переслідуваних цілей, види тестування можна умовно розділити на наступні типи:**

* функціональне тестування **(functional);**

Розглядалось вище

* нефункціональне тестування **(non-functional);**

**Non-functional testing** (Нефункціональне тестування) включає тестування нефункціональних вимог системи, таких як продуктивність, безпека, масштабованість, зручність використання, надійність тощо.

* тестування пов’язане зі змінами.

Це - Retesting і Regression Testing,

**Повторне тестування (Retesting)** — це процес перевірки конкретних тестів, у яких виявлено помилки під час остаточного виконання. На знайдені помилки формуються баг репорти та передаються розробникам для виправлення. Після виправлення баги повертаються тестувальникам для перевірки. Цей цикловий процес називається повторним тестуванням. Для цього не потрібно налаштовувати нове середовище. Повторне тестування виконується з тими самими даними та тим самим середовищем, але з новою збіркою.

**Retesting використовується для:**

* перевірки конкретної помилки чи проблеми, на яку був створений bug report;
* для підтвердження від тестувальника для розробника, що проблема дійсна та її можна відтворити повторно;
* або, навпаки, для підтвердження від розробника, що дефект був виправлений;
* і навіть Retesting використовується для тестування всього модуля або компонента з метою підтвердження роботи очікуваної функціональності.

Якщо простіше, Retesting застосовується для перевірки якості будь-якої конкретної функції, компонента чи модуля програми, якщо в цих частинах були виявлені баги.

**Регресійне тестування (Regression testing)** — це тип тестування функціональності програмного забезпечення після внесення змін на фазі системного тестування або супроводу продукту. Це робиться для того, щоб розуміти, що продукт нормально працює з новими функціями, виправленнями помилок або будь-якими змінами в існуючій функціональності. Та за результатами регресійного тесту можна підтвердити, що зміни не вплинули на працездатність решти функціональності програми або ж спростувати цей факт. Мета регресійного тестування полягає в тому, щоб перевірити, чи нові зміни коду не впливають негативно на існуючі розроблені та протестовані функції програми. Та зменшення кількості багів у системі на момент релізу. Під час виконання регресійного тестування виконуються як функціональні, так і нефункціональні тести.

**Regression testing необхідно проводити:**

* коли до наявної програми додається нова функція;
* при зміні середовища тестування;
* або коли внутрішній код перенесено на іншу платформу;
* якщо на етапі тестування було виявлено багато критичних помилок;
* основні проблеми з продуктивністю та збої у роботі програми усунуті розробниками;
* додані виправлення, які зачіпають більшу частину функціональності системи;
* інтерфейс програми було змінено для покращення взаємодії з користувачем;
* додане покращення бізнесового характеру, яка впливає на основний фукціонал системи.

### **За знанням системи:**

* тестування чорної скриньки **(black box);**
* тестування білої скриньки **(white box);**
* тестування сірої скриньки **(gray box).**

**Методи детально розглядалися раніше**

### **За ступенем автоматизації:**

* ручне тестування **(manual testing);**

**Ручне тестування** — є типом тестування, в якому тестовий випадок виконується вручну людиною.

* автоматизоване тестування **(automated testing);**

**Автоматизоване тестування –** це тип тестування, в якому тестування виконується з використанням різних інструментів автоматизації та скриптів.

* напівавтоматизоване тестування **(semiautomated testing).**

**Напівавтоматизоване тестування** — у цьому випадку частина тесту проробляється вручну, а інша частина за допомогою заздалегідь підготовленого коду (автоматизованого скрипту

### **За ступенем ізольованості компонентів:**

* компонентне (модульне) тестування **(component / unit testing);**
* інтеграційне тестування **(integration testing);**
* системне тестування **(system / end-to-end testing).**

**Методи детально розглядалися раніше**

### **За часом проведення тестування:**

* Альфа-тестування **(alpha testing):**

Це такий тип тестування, який виконується всередині організації-розробника з можливим частковим залученням кінцевих користувачів. Може бути формою внутрішнього приймального тестування. Продукт вже можна показувати зовнішнім користувачам, але він не остаточно дороблений, тому основне тестування виконує організація-розробник.

* тестування нової функціональності **(new feature testing);**
* регресійне тестування **(regression testing);**

Розглядалось вище

* тестування при здачі **(acceptance testing);**

Accessibility Testing — це перевірка сайту або додатку на доступність і простоту користування ним людьми з обмеженими можливостями. Це специфічний для української ментальності вид тестування, але на закордонного замовника за недотримання стандартів і норм законодавства його держави можуть подати до суду.

* Бета-тестування **(beta testing).**

Цей тип тестування виконується за межами організації-розробника з активним залученням кінцевих споживачів або замовника. Може бути формою зовнішнього приймального тестування. Продукт можна відкрито показувати зовнішнім користувачам, він достатньо стабільний. Але проблеми все ще можуть бути, і для їх виявлення потрібен зворотній зв’язок від реальних користувачів. Як приклад можна навести баги, які знаходять користувачі під час бета-тестування комп’ютерних ігор.

### **За ознакою позитивності сценаріїв:**

* позитивне тестування **(positive testing);**

**Happy path testing** — це вид позитивного тестування позитивного, коли у поточний варіанту ми вводимо валідні дані для програми. Підхід зазвичай використовується у автоматизованому тестуванні.

* негативне тестування **(negative testing).**

Rapid Testing або негативне тестування. **Наміренно негативне тестування** — це вид тестування програмного забезпечення, в якому тестувальнику слід оцінити надійність програми, визначити методи, як заставити продукт видавати невірні відповіді, повідомлення про помилки або взагалі перервати дію відмовою при наданні неправильних вхідних даних або даних тесту.

З цією метою тестер застосовує спроби виконання програми:

* на платформах, на яких її виконання не планувалось;
* при відсутності комунікаційних ліній або при вводі неправильних вхідних даних;
* при відсутності файлів даних;
* при відсутності записів в базах даних або при довільно переставлених даних в файлах даних;
* при невірно введених іменах посилань або при невизначених, неправильних або відсутніх конфігураційних параметрах;
* при вимкнених периферійних пристроях типу принтерів, сканерів, зовнішніх дисків, зовнішніх жорстких дисків, зовнішніх динаміків і т. д.
* введення неправильних даних, що можуть бути у формі недопустимих даних, введених користувачем, випадково заповнених даними комунікаційних буферів, недопустимих значень в індексних файлах, переповнення журнальних файлів логів, в яких покажчик знаходиться в кінці файлу і т. д.

*Здатність програми без збою витримати негативне тестування, називається стійкістю програми.*

### **За ступенем підготовленості до тестування:**

* тестування за документацією **(formal testing);**

**Documentation testing (Тестування документації)** — тестування документації передбачає перевірку та валідацію документів та інструкцій користувача системи.

* Ad Hoc Testing (інтуїтивне) тестування **(ad hoc testing).**

**Experience based testing** (Тестування на основі досвіду) — це коли тестові методи та техніки тестування програмного забезпечення повністю базуються на досвіді або інтуїції тестера. Дві найбільш поширені форми тестування на основі досвіду — **Ad-hoc** тестування та **Exploratory testing**.

**Ad-Hoc Testing** — це вид неформального, імпровізованого тестування програмного забезпечення. Синонімами Ad-Hoc тестування є довільне та інтуїтивне тестування. Ad-Hoc тестування виконується без попередньої підготовки до тестування продукту, без визначення очікуваних результатів, проектування тестових сценаріїв тощо. Воно не вимагає ніякої документації, планування, процесів, яких, як правило, слід дотримуватися при виконанні тестування.

Головна перевага, Ad-Hoc тестування часто надає можливість віднайти складні для відтворення і важковловимі дефекти, які неможливо було б знайти, використовуючи стандартні сценарії перевірок.

Проводячи Ad-Hoc тестування, тестувальник покладається на своє загальне уявлення про продукт, порівнюючи його зі схожими продуктами, з власного досвіду. Однак, при Ad-Hoc тестуванні є зміст володіти загальною інформацією про продукт, особливо якщо проект дуже складний і великий.

Найчастіше Ad-Hoc тестування виконується коли власник не оперує конкретними цілями, потребами, вимогами до свого програмного продукту. Звісно у нього не має навіть натяку на щось подібне до проектної документації.

різниця між Exploratory та Ad-Hoc Testing?

Різниця між Ad-Hoc і Exploratory Testing в тому, що теоретично, Ad-Hoc тестування може провести будь-хто, а для проведення Exploratory тестування необхідна фахова майстерність і володіння певними техніками тестування.

**Monkey Testing** — це теж довільний тип тестування, який виконується без чітко визначеного тестового сценарію. Назва є метафоричною, мається на увазі, що операції введення даних є абсолютно випадковими та беззмістовними, наче їх насправді виконує механічна мавпа.

Головна мета Monkey Testing якомога швидше, використовуючи різні варіації вхідних даних, порушити роботу системи або викликати її зупинку «crash» (простими словами — зламати). Наприклад, мавпячий тест може вводити довільні рядки в поля вводу, імітувати одночасне натискання по кнопках, хаотичні скроли тощо.

Для автоматизації Monkey Testing сьогодні існує багато додатків на різних платформах. Хоча Monkey Testing може здійснюватися і людиною з точки зору «неотесаного» користувача.

Різниця між Monkey та Ad-Hoc тестуванням?

У разі проведення Ad-Hoc тестування, хоча й немає заздалегідь визначених або документально підтверджених тестів, тестувальники мають мати деяке розуміння програми. У разі Monkey тестування мавпи-випробувачі не повинні мати ніякого розуміння програми.

тестування Горили?

### **Інші види тестування:**

**Scenario testing (**Тестування сценаріїв) — це тип тестування, в якому тестування складних тестових потоків спрощується шляхом розбиття на частини **(сценарії, потоки, flow).** Такий підхід допомагає перевірити програму від початку до кінця (end-to-end).

**Gorilla Testing** — це тестування окремих модулів або функціональних можливостей програми, щоб перевірити їх надійність. Не має жодного зв’язку із Monkey Testing. Мається на увазі що горила — важка тварина і може щось зламати за 1 підхід. Цей тип тестування перейшов у тестування ПЗ з виробництва електроніки та перевірки її якості.

Fuzz Testing (Fuzzing)?

**Fuzz testing** — це метод «грубої сили» від білих хакерів. Попередник Автоматизованого Тестування та Тестування Безпеки. Суть Fuzz testing надзвичайно проста, полягає в тому що на вхід програми надсилаються масиви випадкових, не валідних і не очікуваних даних — які називаються Fuzz (Фаззери) з метою взлому, падіння, зависання, порушення внутрішньої логіки і перевірок в коді програми, витоку пам’яті.

При цьому про внутрішній устрій програми тестувальнику нічого не відомо. Для цього фреймворки фаззерів, наприклад Sulley (фреймворк для створення складних структур даних) перебирає астрономічну кількість варіантів комбінацій (як неважко здогадатися, повний перебір зростає експоненціально від розміру вхідних даних O (c n), c> 1).

Fuzz testing хороший спосіб перевірити систему, перестрахуватися і виявити у ній слабкі місця до атак вірусів, троянів, шкідливих програм, Dos-атак, SQL injection, Тестування Безпеки взагалом.

**Об’ємне тестування (VolumeTesting)** — це тип тестування продуктивності, який проводиться зі збільшенням не навантаження і часу роботи, а збільшенням кількості оброблюваних даних, які зберігаються і використовуються в програмі. За допомогою VolumeTesting перевіряється масштабованість програми. Ось, таким тестуванням можна ідентифікувати вузьке місце куди не «влізає» задокументований обсяг даних.

**Тестування Spike** — це тип Performance Testing, в якому вимірюється продуктивність програми, коли за імпровізованих умов раптово збільшується кількість активних користувачів під час тестування навантаження.

**Configuration Testing (Тестування конфігурації)** — це спеціальний вид тестування, спрямований на перевірку роботи програмного забезпечення при різних конфігураціях системи (заявлених у документації: платформах, підтримуваних драйверів, при різних конфігураціях комп’ютерів, спеціалізованого обладнання т. п.) …і оцінка результатів від зміни необхідної конфігурації.

**Тестування глобалізацією** — це вид тестування, в якому додаток оцінюється крізь призму придатності його функціонування у всьому світі, в різних культурах, на різних мовах, у певному мовному регіоні чи країнах.

**A / B Testing** — вид тестування в якому два варіанти програмного продукту піддаються впливу кінцевих користувачів а потім аналізується поведінка користувачів у кожному варіанті, і по результатах вибирається та використовується кращий варіант. Хороший інструмент для моніторингу A / B Testing в режимі реального часу однойменний вбудований інструмент в Google Аналітиці.

**Split testing** — те саме що A / B testing.

**Тестування паралельної роботи** — багатокористувацьке тестування, в якому додаток оцінюється шляхом аналізу поведінки програми з одночасною роботою користувачів, які отримують ту саму функціональність.

**All pair Testing** - це вид тестування, в якому додаток перевіряється за усіма можливими комбінаціями змінних значень і вхідних параметрів.

**Failover Testing (Перебійне тестування)** — це тип тестування, який використовується для перевірки здатності додатків виділяти більше ресурсів (більше серверів) у разі відмови та передачі обробної частини в резервну систему.

**Robustness Testing** - **це тестування надійності** (міцність)– методологія забезпечення якості спрямована на тестування надійності програмного забезпечення. Тестування на міцність також було використано для опису процесу перевірки надійності (тобто правильності) тестових випадків у тестовому процесі.

**Penetration testing, Pentesting (Тест на проникнення)** — це популярна у цілому світі послуга з перевірки інформаційної безпеки системи, а також аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту додатків: атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Цей вид тестування безпеки програмного забезпечення полягає в тестуванні різними зовнішніми методами проникнення, які потенційно міг би застосувати хакер.

**Vulnerability testing** (Тести на вразливість) — охоплює тестування та ідентифікацію вразливостей, недоліків в процедурах безпеки системи, розробці, пошук слабких місць у додатку, що може призвести до порушення політики безпеки системи або, іншими словами, отримання несанкціонованого доступу хакерів.

Також виокремлюють таке поняття як **Vulnerability Assessment** — це техніка тестування програмного забезпечення для оцінки раптового підвищення небажаних ризиків, пов’язаних із безпекою системи.

### **Залежить від двох механізмів тестування:**

1. Vulnerability Detection

2. Penetration Testing

**Тестування бази даних (DataBase Testing)** — це вид тестування, яким перевіряємо цілісність Бази Даних і фактичне її відображення на Front-end -і. Тестування Баз Даних передбачає валідацію даних у базі даних, перевірку зв’язків на помилки (нащадків, батьків, запис foreign key to a parent record який був видалений тощо), оновлення записів у Базі Даних.

**Тестування API (API Testing)** — це тестування, яке передбачає тестування програмного інтерфейсу веб-додатків Restful API та веб-сервісів SOAP безпосередньо з використанням клієнта, або інструментів, таких як SOAPUI, Jmeter, Postman.

**Тестування ETL (Extract-Transform-Load — пер. з англ. Вийняти-Перетворити-Завантажити)** — є видом тестування, навіть цілою концепцією, при тестуванні Баз Даних, яка передбачає перевірку логічної послідовності даних після їх вилучення з джерела, на шляху завантаження даних в кінцеву Базу Даних.

**Data warehouse testing** — те ж, що і тестування ETL.

**Пілотне тестування** — це тестування, проведене в якості випробування обмеженою кількістю користувачів, які оцінюють систему та надають свої відгуки про неї до повного розгортання.

**Backend testing** — це вид тестування, який вимагає тестування бекенду системи, що складається з тестування Бази даних та API у додатку.

**End to end testing (2Е2)** — вид тестування, який передбачає тестування програми цілком на усіх рівнях з максимальним покриттям, починаючи **Frondend тестування**, тестування інтерфейсу ПЗ з точки зору кінцевого користувача, закінчуючи **Backend** тестуванням. Тобто проклікування кнопочок, заповнення форм, скролінг до відповідно заданих точок, щоб переконатися що додаток працює відповідно до вимог.

Перевага **End to end тестування** виявляє проблеми в загальній продуктивності системи.

Але оскільки ручному manual тестувальнику у великому додатку таке тестування може зайняти море часу, цей вид тестування чудово автоматизується наприклад, сервісом **Protractor.**

**Recovery testing (Тестування на відновлення)** — це тип нефункціонального тестування, що включає тестування здатності системи відновитись після помилки чи збою у системі. Найпростіший приклад: вимкнули світло чи живлення.

**Compliance testing (Тестування відповідності)** — це тип нефункціонального тестування, який передбачає перевірку того, що побудований продукт відповідає стандартам, процедурами організації та ін. Як правило **Compliance тестування** проводиться зовнішніми компаніями, яке пропонує, продає, доставляє сертифіковане ПЗ певного застосування.

**Conformance testing** — те ж, що і тестування відповідності.

**Forward compatibility testing (**Перехідне тестування сумісності) — передбачає перевірку додатка на новішій версії іншої платформи або програмного забезпечення.

**Backward compatibility testing** (Зворотне тестування сумісності) — навпаки передбачає перевірку додатка на старішій версії іншої платформи або програмного забезпечення.

**Downward compatibility testing & Backward compatibility testing** — те ж саме.

**Destructive testing** — вид тестування, який спрямований на перевірку надійності програми шляхом навмисного порушення цілісності або розчленування програми.

**Fault injection testing (Тестування на несправності)** — це тип випробувань, в якому помилка навмисно вводиться в додаток для поліпшення тестового покриття. **Fault injection testing** — це навіть не вид тестування, а метод тестування, техніка тестування. Багато видів тестування, суть яких вклинювати у систему помилки, наприклад Robutness Testing в першу чергу називається **Fault injection тестуванням.**

**Dependency testing (Тестування залежностей)** — техніка тестування, котра передбачає попередню перевірку вимог до програми на існуючій конфігурації системи, необхідної для правильного функціонування програми. Тестування нових функцій також перевіряється спочатку на існуючій системі. Зрозуміло, отримані результати аналізуються та екстраполюються на потрібну версію.

Питання #53 Поняття тестування інсталяції і деінсталяції?

**Installation testing (Тестування установки)** — вид тестування перевіряє, чи встановлюється програма згідно інструкції з встановлення і так само без проблем видаляється.

**Контрольні питання:**

1.Що таке тестування?

2. Що таке якість програмного забезпечення?

3. Що таке дефект?

4. Назвіть три умови виявлення дефекту.

5. Які існують види тестування залежно від об'єкта тестування? Дайте характеристику кожному.

6. Які існують види функціонального тестування? Дайте характеристику кожному.

7. Які існують види нефункціонального тестування? Дайте характеристику кожному.

8. Які існують види тестування залежно від глибини покриття? Дайте характеристику кожному.

9. Які існують тестові активності? Дайте характеристику кожному.

10. Які існують види тестування залежно від знання коду? Дайте характеристику кожному.

11. Які існують види тестування залежно від ступеня автоматизації? Дайте характеристику кожному.

12. Які існують види тестування залежно від ізольованості компонентів? Дайте характеристику кожному.

13. Які існують види тестування залежно від підготовленості? Дайте характеристику кожному.

14. Які існують види тестування залежно від місця і часу проведення?