**Лабораторна робота № 10-1. Підготовка тестів для перевірки ПЗ – перевірка виконання нефункціональних вимог**

**Мета:** Навчитися формувати перелік тестів для перевірки виконання нефункціональних вимог

**Методичні рекомендації до виконання лабораторної роботи.**

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

|  |  |  |
| --- | --- | --- |
| №№ | Рівень тестування / підрівень / підхід / метод | Види нефункціонального 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 – 17.04.2024**

**Контрольні питання**

**1.** Що таке тестування програмного забезпечення?

2. Які види тестування програмного забезпечення ви знаєте?

3. Які основні принципи тестування програмного забезпечення?

4. Які види тестування програмного забезпечення ви знаєте?

5. Що таке модульне тестування і для чого воно використовується?

6. Які підходи використовуються для тестування взаємодії програмного забезпечення зі зовнішніми системами?

7. Які основні принципи проведення тестування регресії?

8. Які методики тестування використовуються для перевірки відповідності вимогам функціональності програмного забезпечення?

9. Які методики тестування використовуються для перевірки відповідності нефункціональних вимог програмного забезпечення?

**ТЕОРЕТИЧНІ ВІДОМОСТІ**

Пригадаємо минулу ЛР.

**Рівні тестування (Testing levels):**

* Компонентне або Модульне тестування (Component testing or Unit testing)
* Інтеграційне тестування (Integration testing).

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 (System testing)

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 (Acceptance testing)

Методи цього рівня.

* Тестування замовником самостійно.
* Тестування (Аудит) третьою стороною.
* Спільне тестування за сценаріями із замовником.

**Нефункціональні вимоги**

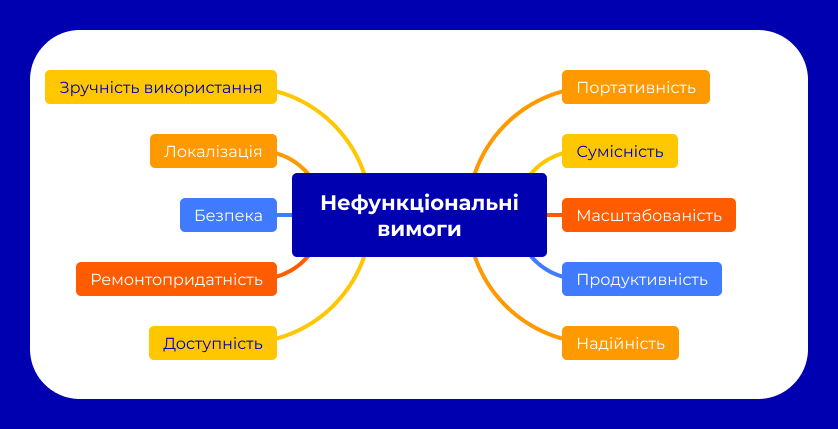
Нефункціональні вимоги або NFR (non-functional requirements) – це набір специфікацій, які описують робочі можливості та обмеження системи та намагаються покращити її функціональність. Це в основному вимоги, які визначають, наскільки добре працюватиме продукт якщо врахувати, наприклад, швидкість, безпеку, надійність, цілісність даних тощо. Структура стандартів ISO/IEC 25000 визначає нефункціональні вимоги як вимоги до якості системи та якості програмного забезпечення.

Як функціональні, так і нефункціональні вимоги описують конкретні характеристики, якими повинен володіти продукт, щоб відповідати потребам зацікавлених сторін і самого бізнесу. Але, як видно з назви, вони зосереджені на різних речах:

* функціональні вимоги визначають, що повинен робити програмний продукт: його характеристики та функції;
* нефункціональні вимоги задають атрибути якості системи.

#### **Основні типи нефункціональних вимог**

Найпоширенішими з них є продуктивність, масштабованість, портативність, сумісність, надійність, доступність, придатність до ремонту, безпека, локалізація, зручність використання тощо.

[](https://training.qatestlab.com/wp-content/uploads/2023/08/banner_qatestlab_tc__Нефункціональні-вимоги_-приклади-типи-підходи__1_2_uk.jpg)

| ***Нефункціональна вимога*** | ***Визначення*** | ***Приклад*** |
| --- | --- | --- |
| Продуктивність | Визначає, наскільки швидко програмна система або її окрема частина реагує на певні дії користувачів за певного навантаження. У більшості випадків цей показник пояснює, скільки часу користувач має чекати, перш ніж відбудеться цільова операція (відображається сторінка, обробляється транзакція тощо) якщо врахувати загальну кількість користувачів на даний момент. | Цільова сторінка, яка підтримує 5000 користувачів на годину, має забезпечувати час відповіді не більше 6 секунд у браузері Chrome настільного комп’ютера з урахуванням відтворення тексту та зображень і через з’єднання LTE. |
| Масштабованість | Оцінює найвищі навантаження, за яких система все ще відповідатиме вимогам продуктивності. | Система повинна бути достатньо масштабованою, щоб підтримувати 1 000 000 відвідувань одночасно та зберігати оптимальну продуктивність. |
| Портативність | Визначає, як система або її елемент можуть бути запущені в тому чи іншому середовищі. Зазвичай містить специфікації обладнання, програмного забезпечення чи іншої платформи використання. Визначає, наскільки добре функції, що виконуються на одній платформі, виконуються на іншій. | Програма, яка працює в Windows 10, повинна працювати в Windows 11 без будь-яких змін у своїй поведінці та продуктивності. |
| Сумісність | Як додатковий аспект портативності, визначає, як система може співіснувати з іншою системою в одному середовищі. Наприклад, програмне забезпечення, встановлене в операційній системі, має бути сумісним із брандмауером або антивірусним захистом. | Програма iOS повинна підтримувати пристрої iPhone, що працюють на версіях ОС: 3.6, 3.3, 3.4, 4.3, 2.3 |
| Надійність | Визначає, наскільки ймовірно, що система або її елемент працюватимуть без збоїв протягом заданого періоду часу за заздалегідь визначених умов. Традиційно цю ймовірність виражають у відсотках. | Система повинна працювати без збоїв у 95 відсотках випадків використання протягом місяця. |
| Ремонтопридатність | Визначає час, необхідний для виправлення дефекта/бага або його компонента, зміни для підвищення продуктивності чи інших якостей або адаптації до мінливого середовища. Придатність до ремонту часто вимірюється за допомогою такого показника, як MTTRS – середній час відновлення системи. | Середній час відновлення системи (MTTRS) після збою не повинен перевищувати 10 хвилин. MTTRS включає весь час коригувального обслуговування та час затримки. |
| Доступність | Описує, наскільки система доступна для користувача в певний момент часу. Виражається як очікуваний відсоток успішних запитів, або як відсоток часу, протягом якого система доступна для роботи протягом певного періоду. | Веб-панель має бути доступною для користувачів із США 99,98% часу щомісяця в робочі години за східним стандартним часом. |
| Безпека | Гарантує, що всі дані всередині системи або її частини будуть захищені від атак шкідливих програм або несанкціонованого доступу. | Шлюз обробки платежів має бути сумісним з PCI DSS. |
| Локалізація | Визначає, наскільки добре система або її елемент відповідає контексту місцевого майбутнього ринку. Контекст включає місцеві мови, закони, валюти, культури, правопис та ін. | Формат дати має бути таким: місяць.число.рік. |
| Зручність використання (Usability) | Наскільки легко клієнту користуватися системою? Зручність використання пропонують оцінювати за п’ятьма параметрами:   * можливість навчання; * ефективність; * запам'ятовуваність; * як часто користувачі роблять помилки? * чи дизайн приємний у використанні? | Рівень помилок користувачів, які надсилають платіжні дані на сторінці оформлення замовлення, не повинен перевищувати 10 відсотків. |

#### **Загальні рекомендації щодо документування нефункціональних вимог**

*Вимоги мають бути такими, щоб їх можна було виміряти та перевірити*. Щоб зрозуміти, чи відповідає система обмеженням якості, необхідно попередньо визначити кількісні вимоги. Потрібно вказати одиниці вимірювання, методи, які будуть використовувати, а також рівні успіху та невдачі.

*Вимоги краще встановлювати до компонентів системи*, а не до цілих продуктів. Якщо користувачі ніколи не взаємодіють з деякою частиною продукту (наприклад, панеллю адміністратора), встановлення обмежень продуктивності для цих компонентів може бути марною, оскільки команда витрачатиме багато зусиль без видимої вигоди.

*Зв'язок NFR із бізнес-цілями.* Хвилинна різниця в доступності системи може не мати суттєвого впливу на продажі, але іноді може означати додаткові тижні розробки. Бізнес-цілі варто розділяти на системні вимоги.

*Обмеження по залученню третіх сторін.* Якщо використовується сторонній API, який повертає дані повільніше, ніж потрібно, команда мало що зможе з цим зробити.

*Врахування архітектурних обмежень.* Застарілі системи можуть накладати обмеження на якість. Хоча рефакторинг застарілого коду можливий, іноді поточну архітектуру потрібно повністю переробити, щоб відповідати деяким вимогам.

*Використання існуючих стандартів та посібників.* Перед початком роботи з вимогами варто ознайомитись з інструкціями програм для iOS або Android тощо.

***Переваги чіткого визначення та тестування нефункціональних вимог:***

* гарантія дотримання правових норм;
* визначення атрибуту якості програмного забезпечення;
* забезпечення надійності, доступності, продуктивності і масштабованості програмного забезпечення;
* побудова політики безпеки системи;
* забезпечення хорошої взаємодії з користувачем, легкість роботи з програмним забезпеченням і мінімізація фактору витрат;
* покращення загальної якості системи;
* підвищення задоволення користувачів;
* краще узгодження з бізнес-цілями;
* зменшення кількості доопрацювань;
* покращення масштабованості і адаптивності системи;
* краще технічне обслуговування системи.

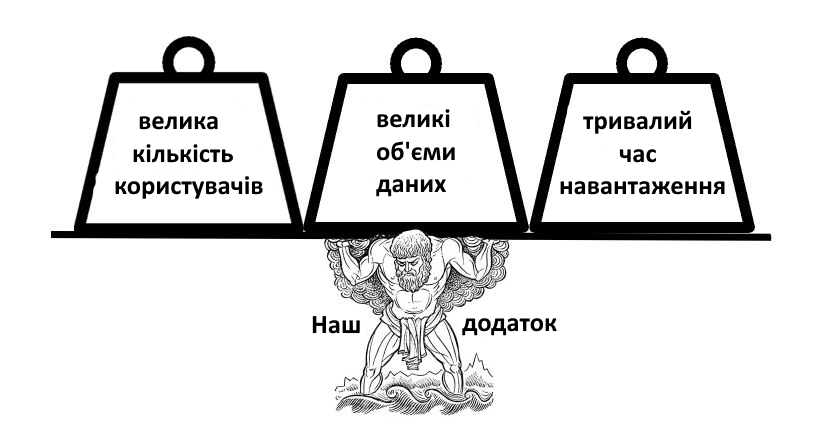
**Недоліки нефункціональних вимог:**

* може впливати на різні підсистеми програмного забезпечення високого рівня;
* збільшення вартості, оскільки вони вимагають особливої ​​уваги на етапі архітектури програмного забезпечення/проєктування високого рівня;
* труднощі з визначенням усіх вимог;
* труднощі з прогнозуванням майбутніх вимог;
* проблеми вимірювання та тестування;
* великі часові затрати, які можуть уповільнити процес розробки;
* ризик зміни вимог: нефункціональні вимоги можуть змінюватися з часом, що може призвести до плутанини;
* конфліктні вимоги: нефункціональні вимоги можуть конфліктувати одна з одною, і може бути важко збалансувати їх та визначити пріоритети для виконання;
* непередбачені вимоги: нефункціональні вимоги можуть бути не повністю визначені на етапі збору вимог, а деякі вимоги можуть бути виявлені лише після розгортання системи.

**Тестування продуктивності**

**Тестування продуктивності** – це комплекс типів тестування, метою якого є визначення працездатності, стабільності, споживання ресурсів та інших атрибутів якості додатка в умовах різних сценаріїв використання та навантажень. Тестування продуктивності дозволяє знаходити можливі вразливості та недоліки в системі з метою запобігання їх негативному впливу на роботу програми в умовах використання. Необхідні параметри роботи системи в певному середовищі можна тестувати за допомогою:

* + Визначення робочої кількості користувачів додатку.
  + Вимірювання часу виконання різних операцій системи.
  + Визначення продуктивності додатка при різних рівнях навантаження.
  + Визначення допустимих меж продуктивності програми при різних рівнях навантаження.

[](https://qalight.ua/wp-content/uploads/2016/09/perfor_test-1.png)

Залежно від характеристик, які нам потрібно протестувати, тестування продуктивності поділяється на типи:

* + - Тестування навантаження (Load Testing) – тестування часу відповіді програми на запити різних типів, з метою переконатися, що додаток працює відповідно до вимог при звичайному користувацькому навантаженні.
    - Стрес-тестування (Stress Testing) – тестування працездатності додатку при навантаженнях, що перевищують користувацькі в декілька разів. При стрес-тестуванні (найчастіше, тільки при ньому) ми можемо отримати реальні дані границь продуктивності додатка, досліджувати здатність програми обробляти виключення, її стабільність та стійкість. Саме в значно збільшених навантаженнях на додаток і полягає різниця між тестуванням продуктивності та стрес тестуванням.
    - Тестування стабільності або робота на відмову (Stability/Reliability Testing) досліджує працездатність додатку при тривалій роботі у часі за нормального навантаження програми.
    - Об’ємне тестування (Volume Testing) – тестування проводиться із збільшенням кількості використовуваних даних, які зберігаються та використовуються в додатку.

Дуже часто при визначенні тестування продуктивності та його типів приходять до помилкового розуміння та плутанини даних термінів. Отже, тестування продуктивності – це перевірка таких нефункціональних вимог, як продуктивність та працездатність програми при різних навантаженнях і умовах. Залежно від досліджуваної характеристики програми, ми можемо виділити такі типи тестування:

* + Навантажувальне тестування (продуктивність при нормальних умовах).
  + Стрес-тестування (працездатність, продуктивність і характеристики додатку при екстремальних навантаженнях).
  + Тестування стабільності (при тривалій роботі).
  + Об’ємне тестування (при збільшених обсягах оброблюваних даних).