**Лекція 13. Види тестування**

Знання видів тестування – це необхідна навичка тестувальника, вона дає розуміння підходів щодо тестування програмного забезпечення, допомагає добре спланувати тестування, а також класифікувати знайдені помилки.

**Найбільш поширені види тестування.**

* *Функціональне тестування* (*на відповідність* або тестування *коректності)* та нефункціональне тестування
* *Тестування безпеки* (Security testing)
* *Структурне тестування*
* *Тестування зручності застосування (ергономічності)*
* *Тестування технічних характеристик*
* *Тестування на надійність.*
* *Тестування продуктивності (Performance testing)* види:
  + *навантажувальне тестування (load testing)*
  + *тестування на стійкість (stress testing*);
  + *тестування обсягу (volume testing)* .
* *Тестування конфігурації (Configuration testing).*
* *Порівняльне тестування ( Back-to-back testing).*
* *Тестування відновлення (Recovery testing).*
* *Керована тестами розробка ( Test-driven development)*
* *Системне тестування*

Відповідно до видів тестування обираються методи до проектування тестів. Нижче надається огляд для попереднього ознайомлення.

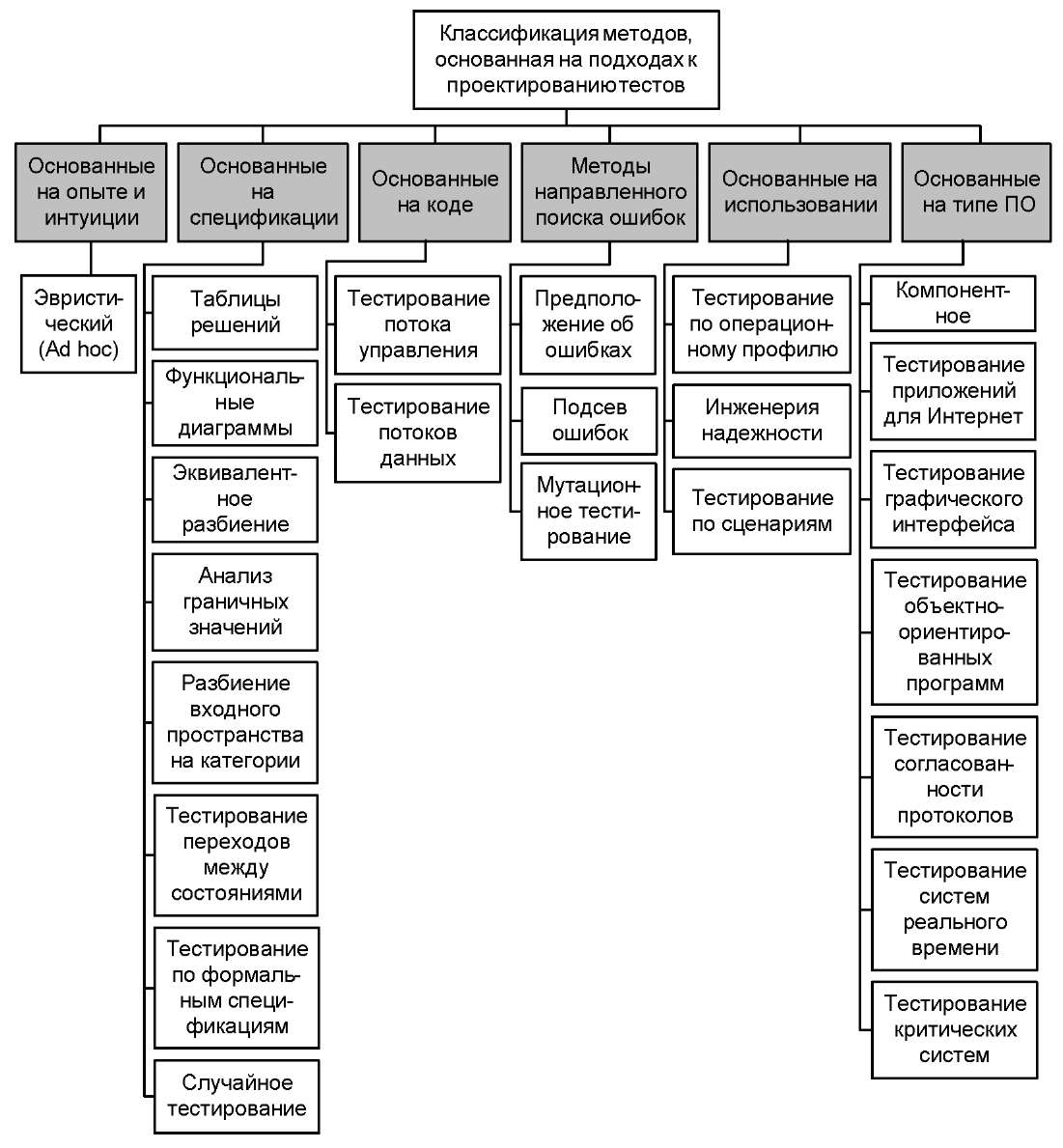


Рисунок 1 - Методи тестування

Тестування правильності. *Мета* – підтвердити, що функції, описані у специфікації вимог до ПЗ, відповідають очікуванням замовника.

Підтвердження правильності ПЗ виконується за допомогою тестів «чорного ящика», що демонструють відповідність вимогам. При виявленні відхилень від специфікації вимог створюється список недоліків. Як правило, відхилення і помилки, виявлені при підтвердженні правильності, вимагають зміни термінів розробки продукту.

**Функціональне та нефункціональне тестування**

Функціональне та нефункціональне тестування перевіряє різні аспекти програмного забезпечення. Функціональне тестування під час тестування програмного забезпечення пов’язане з тим, чи кожна функція програмного забезпечення чи програми відповідає необхідним специфікаціям. З іншого боку, нефункціональне тестування вимірює, наскільки добре працює програмне забезпечення чи програми, а не те, чи вони взагалі працюють.

**Типи функціонального тестування**

Визначення функціонального тестування міститься в його назві. Функціональне тестування стосується перевірки того, чи програмне забезпечення працює належним чином і відповідає очікуванням користувача.

Розглянемо основні типи функціонального тестування:

#### **Модульне тестування**

Розробники або тестувальники використовують модульне тестування, щоб визначити, чи відповідають окремі компоненти або модулі програмного забезпечення чи програми функціональним вимогам. Модульне тестування гарантує належну роботу найменших функціональних частин програмного забезпечення.

Один з видів -тестування покриття коду, яке є життєво важливим. Крім того, також може виникнути потреба виконати модульне тестування для покриття лінії, покриття шляху коду та покриття методу.

##### Переваги модульного тестування:

* Визначення загальної якості коду
* Пошук програмних помилок
* Наявність документації для кращого розуміння інтерфейсу

##### Недоліки модульного тестування:

* Потрібно написати більше коду
* Може свідчити про необхідність змінити конструкцію конструкції
* Не виявляє кожну помилку

#### **Випробування диму (димове тестування)**

Розробники (і іноді тестери) виконують димові тести після кожної нової збірки, щоб забезпечити стабільність і перевірити критичні функції. Випробування на дим перевіряє стабільність усієї системи. Наприклад, ви можете провести димове тестування функціональності сторінки входу або визначити функціональність додавання, оновлення чи видалення записів у новій збірці.

##### Переваги димових тестів:

* Забезпечує стабільність програмного забезпечення для більш детального тестування
* Переконується, що програмне забезпечення не містить аномалій або помилок

##### Недоліки тестування на дим:

* Це не детальне тестування
* Невелика кількість випадків тестування диму може не виявити всіх критичних проблем

#### **Тестування на осудність (перевірка після внесення змін)**

Тестери зазвичай проводять тести на осудність після тестів на дим. Перевірка працездатності гарантує, що конкретні нові функції зі складання чи виправлення помилок програми чи програмної системи працюють належним чином. Якщо тестування диму надихнуло виправлення помилок, перевірка працездатності – це те, де ви зможете визначити, чи спрацювали ці виправлення помилок. Наприклад, якщо димовий тест виявив проблеми входу, перевірка працездатності перевірить виправлення помилок входу та піде на крок далі, щоб переконатися, що нові логіни відповідають усім критеріям входу.

##### Переваги тестів на осудність:

* Економить час, оскільки зосереджується на певних областях функціональності після незначних змін
* Допомагає визначити залежні відсутні компоненти після незначних змін

##### Недоліки тестування на осудність:

* Коротко і не детально
* Вказує лише на те, що зміни спрацювали належним чином

#### **Регресійне тестування (тестування змін)**

Тестувальники використовують регресійне тестування серед типів функціонального тестування, щоб переконатися, що будь-який новий код, оновлення чи виправлення помилок не порушують існуючу функціональність програми чи програмного забезпечення та не викликають нестабільність незалежних функцій. Наприклад, ви не хочете, щоб ваш новий код або виправлення помилок порушили можливість введення даних у старішу частину програмного забезпечення.

При тестуванні змін в системі дуже важливо зрозуміти різницю та межу між поняттями регресійне тестування (Regression testing) та повторне тестування (Retesting).

Регресійне тестування (Regression testing) проводиться з метою перевірки працездатності функціоналу, що існує, та перевірки на відсутність сторонніх помилок після оновлення білда (внесення правок або доповнень в систему).

Повторне тестування (Retesting) – проводиться для підтвердження виправлення помилки та роботи даного функціоналу.

|  |  |
| --- | --- |
| **Regression testing** | **Retesting** |
| Регресійне тестування виконується тільки при додаванні нової фічі (додаткова функціональність ПЗ) або істотній зміні функціоналу системи. | Ретест виконується в тому ж оточенні й з тими ж даними, але на новому білді. |
| Регрес можна проводити паралельно з повторним тестуванням. | Повторне тестування має вищий пріоритет та має бути виконано до регресійного. |
| Тест-кейси можуть бути автоматизовані. | Тест-кейси не можуть бути автоматизовані. |
| В рамках регресійного тестування тест-кейси, які були відмічені раніше як «Passed», повинні бути перевірені повторно. | В рамках повторного тестування (ретест) перевіряються тест-кейси тільки зі статусом «Failed». |

##### Переваги регресійного тестування:

* гарантує, що виправлення чи зміни програмного забезпечення не впливають на те, як старі частини програмного забезпечення працюють із тією самою функціональністю
* гарантує, що проблеми, які ви раніше вирішували, не виникнуть знову
* підтверджує відсутність багів після додавання нової властивості або правки коду;
* може бути виконано з використанням інструментів автоматизації;
* допомагає поліпшити якість продукту.

##### Недоліки регресійних тестів:

* якщо ви не зможете автоматизувати процес, регресійне тестування може бути дорогим і трудомістким, оскільки вам потрібно робити це для кожної невеликої зміни в коді
* для складних сценаріїв необхідні складні тестові випадки

***Повторне тестування (Retesting)***

*Переваги:*

* підтверджує виправлення помилки й коректну роботу функціонала;
* підвищує загальну якість продукту;
* вимагає менше часу на верифікацію;
* не вимагає яких-небудь нових налаштувань середовища тестування.

*Недоліки:*

* тест-кейси для повторного тестування можуть бути виявлені тільки після першого раунду тестування;
* тест-кейси для повторного тестування не можуть бути автоматизовані;
* вимагає додаткового часу для проходження вже пройдених раніше тест-кейсів.

#### **Інтеграційне тестування**

Розробники (і іноді тестери) проводять інтеграційне тестування, щоб переконатися, що окремі модулі в програмному забезпеченні чи програмі підключені та працюють разом. Тестування інтеграції передбачає перевірку логіки та відтворених значень. Він гарантує, що модулі добре інтегруються з інструментами сторонніх розробників, і точно визначає дефекти в обробці винятків. Наприклад, ви можете перевірити, чи сторінка входу приведе вас до правильного модуля після входу. Або ви можете перевірити, чи видалені елементи потрапляють у кошик після видалення.

##### Переваги інтеграційних тестів:

* Забезпечує незалежний процес тестування з охопленням коду всієї системи
* Виявляє помилки або проблеми безпеки на початкових етапах функціонального тестування програмного забезпечення, щоб заощадити час

##### Недоліки інтеграційного тестування:

* Важко виконати
* Забирає багато часу

#### **Бета/тестування юзабіліті (зручності користування)**

Після виконання інших типів функціонального тестування бета-тестування/тестування зручності використання дозволяє фактичним клієнтам перевірити, чи нове оновлення продукту працює належним чином, перш ніж воно стане доступним для всіх. Клієнти залишають відгуки про те, наскільки добре працює оновлення, а розробники розглядають подальші зміни коду для зручності використання. Наприклад, якщо зовнішній вигляд інтерфейсу користувача змінюється з оновленням, бета-тестування дозволяє клієнтам надати відгук про те, що працює, а що ні, і яких функцій не вистачає.

##### Переваги юзабіліті-тестів:

* Оцінює, що подумає кінцевий користувач про зміни, і визначає, чого не вистачає або що не працює при регулярному використанні
* Покращує якість продукту та знижує ризик відмови продукту або ризик незадоволеності клієнтів після запуску

##### Недоліки юзабіліті-тестування:

* Розробники не контролюють процес тестування
* Труднощі з копіюванням помилок досвід бета-тестерів

#### **Структурне тестування**

Структурне тестування направлено на тестування структури системи або компонента. Цей вид тестування, як правило, відносять до тестування «білого» та «сірого» ящиків, оскільки ми перевіряємо, що відбувається всередині системи або додатка.

Методи структурного тестування:

1. ***Строкове покриття (Statement Coverage)*** – перевірка застосування усіх операторів в програмі на використання (хоча б один раз).
2. ***Покриття шляху (Path Coverage)*** – метод тестування, призначений для задоволення критеріїв охоплення кожного логічного шляху через програму.
3. ***Покриття рішення (Branch Coverage)*** – перевіряє, чи має кожна умова розгалуження для програми дійсні чи хибні значення;
4. ***Покриття умови (Condition Coverage)*** – метод, схожий на Branch Coverage, основна відмінність полягає в перевірці стану покриття для умовних і неумовних гілок.

*Переваги:*

* можливість виявити та видалити «зайвий» код;
* можливість виявлення потенційних помилок на ранній стадії;
* забезпечує більш ретельне тестування ПЗ;
* не потребує високих витрат людино-годин.

*Недоліки:*

* вимагає знання коду та інструментів тестування.

**Тестування стабільності та надійності** (Stability / Reliability Testing) — один з видів автоматизованого тестування ПЗ, ціллю якого є перевірка працездатності додатку при тривалому тестуванні з очікуваним рівнем навантаження.

Тестування стабільності та надійності важливе, оскільки воно допомагає:

* Виявити витоки пам'яті та інші проблеми з управлінням ресурсами.
* Гарантувати, що програмне забезпечення здатне витримувати очікуване навантаження протягом тривалого часу.
* Запобігти збоям і перевантаженню системи.
* Підвищити загальну якість і надійність програмного забезпечення.

Тестування стабільності та надійності зазвичай проводиться шляхом запуску програмного забезпечення на тривалий період часу з очікуваним рівнем навантаження. Під час тестування ретельно відстежуються такі показники, як використання ресурсів, швидкість обробки даних та час відповіді додатку.

***Тестування безпеки***.

Комп'ютерні системи дуже часто є мішенню незаконного проникнення. Під проникненням розуміється широкий діапазон дій: спроби хакерів проникнути в систему зі спортивного інтересу, помста колишніх адміністраторів системи тощо.

Тестування безпеки перевіряє фактичну реакцію захисних механізмів, вбудованих в систему, на проникнення.

У ході тестування безпеки випробувач грає роль взламника. Йому дозволено все:

* спроби дізнатися пароль за допомогою зовнішніх засобів;
* атака на систему за допомогою спеціальних утиліт, які аналізують захист;
* цілеспрямоване введення помилок в надії проникнути в систему в ході відновлення;
* перегляд несекретних даних в надії знайти ключ для входу в систему.

Звичайно, при необмеженому часі і ресурсах вдале тестування безпеки зламає будь-яку систему. Завдання проектувальника системи – зробити ціну проникнення більш високою, ніж ціна одержаної в результаті інформації.

Важливим елементом підтвердження правильності є перевірка конфігурації ПЗ. *Конфігурацією ПЗ* називають сукупність усіх елементів інформації, що виробляються в процесі конструювання ПЗ.

*Мінімальна конфігурація ПЗ* включає наступні базові елементи:

1. системну специфікація;
2. план програмного проекту;
3. специфікацію вимог до ПЗ; працюючий або паперовий макет;
4. попереднє керівництво користувача;
5. специфікація проектування;
6. лістинги вихідних текстів програм;
7. план і методику тестування; тестові варіанти і отримані результати;
8. керівництва по роботі та інсталяції;
9. ехе-код виконуваної програми;
10. опис бази даних;
11. керівництво користувача по налаштуванню;
12. документи супроводу; звіти про проблеми ПЗ; запити супроводу; звіти про конструкторських змінах;
13. стандарти та методики конструювання ПЗ.

Перевірка конфігурації гарантує, що всі елементи конфігурації ПЗ правильно розроблені, враховані і достатньо деталізовані для підтримки етапу супроводу в життєвому циклі ПС.

Розробник не може передбачити, як замовник буде реально використовувати ПЗ. Для виявлення помилок, які здатний знайти тільки кінцевий користувач, використовують процес, що включає альфа-і бета-тестування.

Альфа-тестування проводиться замовником в організації розробника. Розробник фіксує всі виявлені замовником помилки і проблеми використання.

Бета-тестування проводиться кінцевим користувачем в організації замовника. Розробник у цьому процесі участі не бере. Фактично, бета-тестування – це реальне застосування ПЗ у середовищі, яке не керується розробником. Замовник сам записує всі виявлені проблеми і повідомляє про них розробнику. Бета-тестування проводиться протягом фіксованого терміну (близько року). За результатами виявлених проблем розробник змінює ПЗ і тим самим готує продукт повністю на базі замовника.

Системне тестування. Даний вид тестування передбачає вихід за рамки області дії програмного проекту і проводиться не тільки програмним розробником. Класична проблема системного тестування – виявлення причини. Вона виникає, коли розробник одного системного елемента звинувачує розробника іншого елемента в причині виникнення дефекту. Для захисту від подібного звинувачення розробник програмного елемента повинен:

1. передбачити засоби обробки помилки, які тестують всі входи інформації від інших елементів системи;
2. провести тести, моделюючі невдалі дані або інші потенційні помилки інтерфейсу ПЗ;
3. записати результати тестів, щоб використовувати їх як доказ невинуватості у випадку «зазначення причини»;
4. взяти участь в плануванні та проектуванні системних тестів, щоб гарантувати адекватне тестування ПЗ.

*Системні тести повинні перевіряти, що всі системні елементи правильно об'єднані і виконують призначені функції.* Розглянемо основні типи системних тестів – тестування відновлення, тестування безпеки, стресове тестування та тестування продуктивності/ працездатності .

*Тестування відновлення*.

Комп'ютерні системи повинні відновлюватися після відмов і відновлювати обробку в межах заданого часу. У деяких випадках система повинна бути відмовостійкою, тобто відмови обробки не повинні бути причиною припинення роботи системи. В інших випадках системна відмова має бути усунута в межах заданого кванта часу, інакше замовнику наноситься серйозний економічний збиток.

Тестування відновлення використовує самі різні шляхи для того, щоб змусити ПЗ відмовити, і перевіряє повноту виконаного відновлення. При автоматичному відновленні оцінюються правильність повторної ініціалізації, механізми копіювання контрольних точок, відновлення даних, перезапуск. При ручному відновленні оцінюється, чи знаходиться середній час відновлення в допустимих межах.

*Стресове тестування*.

Стресові тести проектуються для нав'язування програмами ненормальних ситуацій. По суті, проектувальник стресового тесту запитує, як сильно можна розхитати систему, перш ніж вона відмовить?

Стресове тестування проводиться при ненормальних запитах на ресурси системи (за кількістю, частотою, розміром-об'ємом).

Приклади:

* генерується 10 переривань в секунду (при середній частоті 1,2 переривання в секунду);
* швидкість введення даних збільшується прямо пропорційно їх важливості (щоб визначити реакцію вхідних функцій);
* формуються варіанти, що вимагають максимуму пам'яті та інших ресурсів;
* генеруються варіанти, що викликають переповнення віртуальної пам'яті;
* проектуються варіанти, що викликають надмірний пошук даних на диску.

Різновид стресового тестування називається *тестуванням чутливості*. У деяких ситуаціях (зазвичай в математичних алгоритмах) дуже малий діапазон даних, що міститься в межах правильних даних системи, може викликати помилкову обробку або різке зниження продуктивності. Тестування чутливості виявляє комбінації даних, які можуть викликати нестабільність або неправильність обробки.

### ****Типи нефункціонального тестування****

Після визначення того, чи програмне забезпечення робить те, що повинно, нефункціональне тестування може виміряти, наскільки добре воно працює за різних обставин.

Якщо в рамках функціонального тестування ми відповідаємо на питання «Чи працює система?», то нефункціональне відповідає на питання: «Як добре працює система?». Нефункціональне тестування направлено на перевірку тих аспектів ПЗ, які можуть бути описані в документації, але не відносяться до функцій програмних продуктів.

Нефункціональне тестування складається з підвидів:

1. ***Тестування стабільності*** – ***Stability testing*** – перевірка працездатності додатку при тривалому тестуванні з очікуваним рівнем навантаження.
2. ***Юзабіліті тестування*** – ***Usability testing*** – дослідження для визначення зручності використання ПЗ.
3. ***Тестування ефективності*** – ***Efficiency testing*** – перевірка необхідних обсягів коду і ресурсів QA,  що використовуються програмою для виконання окремої функції.
4. ***Тестування ремонтопридатності*** – ***Maintainability testing*** – цей підвид нефункціонального тестування визначає наскільки легко підтримувати працездатність системи.
5. ***Перевірка портативності*** – ***Portability testing*** – тестування доступності перенесення окремого компонента або всього програмного забезпечення з одного оточення на інше (Windows 8.1 -> Windows 10, Windows -> MacOS).
6. ***Тестування «пра-витоків»*** – ***Baseline testing*** – перевірка документації та специфікації, за якою будуть написані тест-кейси. До цього підвиду тестування можна віднести й тестування вимог.
7. ***Приймальне тестування*** – ***Compliance/Acceptance testing*** – перевірка продукту на відповідність критеріям готовності.
8. ***Тестування документації*** – ***Documentation testing*** – перевірка всієї створеної в рамках тестування документації (від майстер тест-плану до тест-кейсів).
9. ***Тестування витривалості системи*** – ***Endurance testing*** – тестування системи при високому навантаженні протягом тривалого періоду часу з метою вивчення її поведінки.
10. ***Тестування навантаження*** – ***Load testing*** – як правило, проводиться з метою визначення поведінки ПЗ під очікуваним рівнем навантаження.
11. ***Тестування продуктивності*** – ***Performance testing*** – перевірка швидкості роботи ПЗ або його окремих функцій.
12. ***Тестування сумісності*** – ***Compatibility testing*** – тестування системи під час роботи в різних середовищах: «залізо», софт частина тощо.
13. ***Тестування безпеки*** – ***Security testing*** – проводиться для відповіді на питання «Чи є додаток безпечним/захищеним чи ні?».
14. ***Об'ємне тестування*** – ***Volume testing*** – тестування ПЗ з використанням баз даних певного розміру.
15. ***Стрес тестування*** – ***Stress testing*** – це тестування в обмежених умовах, наприклад, перевірка поведінки системи (відсутність крешів) за умов нестачі ресурсів комп’ютера (оперативної пам’яті або місця на HDD/SSD дисках).
16. ***Тестування швидкості відновлення*** – ***Recovery testing*** – проводиться з метою визначення швидкості відновлення системи у разі софтверного креша (падіння програмного забезпечення) або помилки «заліза».
17. ***Тестування локалізації, інтернаціоналізація*** – ***Localization testing*** – перевірка ПЗ на відповідність мовних, культурних та/або релігійних норм. Локалізація – перевірка відображення усіх перекладених текстів програмного забезпечення.

***Тестування продуктивності***.

У системах реального часу та вбудованих системах неприпустимо ПЗ, яке реалізує необхідні функції, але не відповідає вимогам продуктивності.

Тестування продуктивності перевіряє швидкість роботи ПЗ в комп'ютерній системі. Продуктивність тестується на всіх кроках процесу тестування. Навіть на рівні елемента при проведенні тестів «білого ящика» може оцінюватися продуктивність індивідуального модуля. Тим не менш, поки всі системні елементи не об'єднаються повністю, не може бути встановлена дійсна продуктивність системи. Іноді тестування продуктивності поєднують зі стресовим тестуванням. При цьому нерідко потрібен спеціальний апаратний і програмний інструментарій. Наприклад, часто потрібно точне вимірювання використовуваного ресурсу (процесорного циклу і т. д.). Зовнішній інструментарій регулярно відстежує інтервали виконання, реєструє події (наприклад, переривання) і машинні стану. За допомогою інструментарію випробувач може виявити стани, які призводять до деградації і можливих відмовлень системи.

Тестування продуктивності дозволяє розробникам дізнатися, наскільки добре працюють програмні компоненти. Він вимірює загальну якість програмного забезпечення, щоб визначити його швидкість і масштабованість.

Деякі приклади тестування продуктивності включають вимірювання часу відгуку, пошук вузьких місць і точок, де програмне забезпечення дає збій.

##### Переваги продуктивних тестів:

* Визначення швидкості програмного забезпечення
* Оптимізація програмного забезпечення
* Визначення вантажопідйомності для задоволення кінцевих користувачів

##### Недоліки тестування продуктивності:

* Може бути дорогим
* Потрібні кілька пристроїв у кількох місцях, щоб визначити, які труднощі можуть виникнути у споживачів

#### **Тестування навантаження**

Навантажувальне тестування допомагає розробникам зрозуміти, як система працює за очікуваних навантажень і максимальних навантажень. Навантажувальне тестування допомагає переконатися, що програмне забезпечення працює та відповідає очікуванням користувачів за нормальних навантажень, а не тестувати його за екстремальних можливостей.

##### Переваги навантажувального тестування:

* Виявляє звичайні вузькі місця
* Визначає ідеальну інфраструктуру для регулярного використання
* Зменшує час простою під час звичайних стрибків трафіку

##### Недоліки навантажувальних тестів:

* Виявлення недоліків у вантажопідйомності, усунення яких може бути дорогим
* Виявлення обмежень, які можуть підштовхнути потенційних користувачів до інших варіантів під час великого трафіку

#### **Стрес-тестування**

Стрес-тестування показує, наскільки добре програмна система працює в умовах найвищого навантаження. Наприклад, ви можете перевірити, як працює система, коли в систему входить більше клієнтів, ніж зазвичай.

##### Переваги стрес-тестів:

* Виявлення того, як система працюватиме після збою та наскільки добре вона відновлюватиметься
* Надання картини того, як система буде працювати за регулярних і нестандартних умов
* Надає уявлення про те, чи потрібне обмеження користувача

##### Недоліки стрес-тестування:

* Знати, як писати сценарії стрес-тестування для всіх потенційних сценаріїв
* Витратний і важкий для виконання вручну

***Порівняльне тестування*** (Back—to—back testing)

Одиничний набір тестів, що дозволяють порівняти дві версії системи.

**Тестування на відмову та відновлення (Failover and Recovery Testing)** – вид тестування, основною місією якого є визначення здатності ПЗ до опору та відновленню після збоїв у роботі, які виникли як всередині програми, так і від інших програмно-незалежних факторів (апаратура, мережа і т.д.).

Даний вид тестування має досить специфічний (порівняно з іншими видами) підхід до виконання тестів, так як об’єктами дослідження є:

* Поведінка ПЗ при перериванні обробки даних.
* При втраті мережі.
* При відключенні електроенергії (на стороні клієнта або сервера).
* При втраті підключення носіїв даних.

Отже і різні сценарії тестування розробляються спираючись на вищезгадані фактори впливу на здатність ПЗ до відновлення після збою.

Тестування на відмову та відновлення особливе актуально при розробці систем, які повинні працювати протягом тривалого часу, аж до 24/7. Від здатності такого ПЗ відновлювати працездатність після непередбаченої ситуації, а також мінімізувати втрати даних буде залежати не тільки репутація компанії-розробника, а часом і щось більше, ніж гроші.

При моделюванні ситуації збою, оцінюється як ступінь втрати даних (чи знаходиться вона в межах допустимого), так і здатність системи протоколювати всі транзакції та статус їх виконання.

**TDD** — **Test**-**Driven Development** **TDD** — технологія розробки програмного забезпечення, яка використовує короткі ітерації розробки, що починаються з попереднього написання тестів, які визначають необхідні покращення або нові функції. Кожна ітерація має на меті розробити код, який пройде ці тести. Нарешті, програміст або група вдосконалюють код для погодження змін. Один із ключових моментів TDD полягає у тому, що підготовка тестів перед написанням самого коду пришвидшує процес внесення змін. Варто зауважити, що керована тестами розробка є методологією розробки програмного забезпечення, а не його тестування.

Test-Driven Development відноситься до концепції екстремального програмування, яка стверджує, що спершу потрібно писати тести, а вже потім код, яка веде свій початок з 1999 року,[]](https://uk.wikipedia.org/wiki/%D0%9A%D0%B5%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D1%82%D0%B5%D1%81%D1%82%D0%B0%D0%BC%D0%B8_%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0#cite_note-Cworld92-1) однак, останнім часом спостерігається загальніший інтерес до даної методології. Програмісти також використовують дану методологію для вдосконалення і налагодження первинного / сирцевого коду, раніше написаного з використанням інших методологій розробки.

**Системне тестування**

Системне тестування якісно відрізняється від інтеграційного і модульного рівнів. Системне тестування розглядає тестову систему в цілому і оперує на рівні користувацьких інтерфейсів, на відміну від останніх фаз інтеграційного тестування, яке оперує на рівні інтерфейсів модулів. Різні і цілі цих рівнів тестування. На рівні системи часто важко та малоефективно аналізувати проходження тестових траєкторій всередині програми або відслідковувати правильність роботи конкретних функцій.

Основне завдання системного тестування — виявлення дефектів, пов’язаних з роботою системи в цілому:

* відсутня або неправильна функціональність;
* невірне використання ресурсів системи;
* непередбачені комбінації даних користувацького рівня;
* несумісність з оточенням;
* непередбачені сценарії використання;
* незручність у застосуванні тощо.

Системне тестування проводиться над проектом в цілому за допомогою методу «чорного ящика». Структура програми не має ніякого значення, для перевірки доступні тільки входи і виходи, видимі користувачеві. Тестуванню підлягають коди і користувацька документація.

**Категорії тестів системного тестування**

1. Повнота вирішення функціональних завдань.

2. Стресове тестування — на граничних обсягах навантаження вхідного потоку.

3. Коректність використання ресурсів (витік пам’яті, повернення ресурсів).

4. Оцінка продуктивності.

5. Ефективність захисту від перекручування даних і некоректних дій.

6. Перевірка інсталяції і конфігурації на різних платформах.

7. Коректність документації.

Інший приклад поділу на категорії:

* Функціональне тестування (functional testing).
* Тестування продуктивності (performance testing).
* Стресове тестування (stress testing).
* Навантажувальне тестування (load testing).
* HP LoadRunner (комерційне ПЗ для тестування).
* Тестування зручності використання (usability testing).
* Тестування інтерфейсу користувача (UI testing).
* Тестування безпеки (security testing).
* Тестування локалізації (localization testing).
* Тестування сумісності (compatibility testing).

Оскільки системне тестування проводиться на користувацьких інтерфейсах, створюється ілюзія того, що побудова спеціальної системи для автоматизації тестування не завжди необхідна. Однак обсяги даних на цьому рівні такі, що зазвичай більш ефективним підходом є повна або часткова автоматизація тестування, що призводить до створення тестової системи набагато більш складною, ніж система тестування, що застосовується на рівні тестування модулів або їх комбінацій.

**Питання для самоконтролю:**

**1.** Назвіть рівні тестування (Test Levels).

**2.** Над чим проводяться тести?

**3.** Навести типові цілі тестування (Objectives for testing).

**4.** Приймальне тестування (Acceptance / qualification testing)

**5.**Пояснити, навіщо використовують альфа і бета-тестування (Alpha and beta testing), хто його проводить, навести приклади.

**6.** Функціональні тести / тести відповідності (Conformance testing / Functional testing / Correctness testing).

**7.** Проаналізувати процес регресійного тестування (Regression testing).

**8.** Проаналізувати процес навантажувального тестування (Stress testing).

**9.**Проаналізувати процес порівняльного тестування (Back-to-back testing), обґрунтувати доцільність його проведення.

**10.** Тестування спроможності системи на відновлення (Recovery testing).

**11.** Проаналізувати процес конфігураційного тестування (Configuration testing), обґрунтувати доцільність і умови його проведення.

**12.** Описати основні етапи тестування зручності і простоти використання (Usability testing), обґрунтувати доцільність проведення такого тестування.

**13** У чому полягає відмінність інтеграційного і модульного видів тестування від системного?

**14** Що саме аналізує системне тестування?

**15** Який метод тестування застосовується під час тестування системи в цілому?

**16** На які категорії можна поділити тести системного тестування?

**17** Які частини програмного комплексу тестуються в кожному з названих випадків?

**18.** Що таке регресійне тестування?

**19.** Опишіть порядок дій під час виконання регресійного тестування.

**20.** З якою метою можуть комбінуватися рівні тестування? Чого можна досягти за рахунок комбінування?