**Лекція 11 Моделі вимірювання та тестування програмного коду ІС. Загальний огляд видів тестування**

*Якість програмного забезпечення* ***(Software quality)*** асоціація IEEE визначає як ступінь відповідності програмного забезпечення встановленій комбінації властивостей. У Міжнародному стандарті якості ISO 9000:2007 це поняття визначається як сукупність характеристик ПЗ, які забезпечують встановлені та очікувані вимоги.

До характеристик якості ПЗ відносять (рис. 1):

* *функціональність* – виконання заявлених функцій, відповідність стандартам, функціональна сумісність, безпека, точність;
* *надійність* – стійкість до відмов, можливість відновлення, завершеність;
* *ефективність* – економія часу, ефективність використання;
* *зручність у використанні* – ергономічність, інтуїтивна зрозумілість, повна документація;
* *зручність для супроводу* – стабільність, придатність для контролю та внесення змін;
* *портативність* – зручність установки, можливість заміни, сумісність.

Якість ПЗ

Функціональність

Надійність

Зручністьу використанні

Ефективність

Зручність супроводу

Портативність

Рисунок 1 – Характеристики якості ПЗ

*Забезпечення якості* ***(Quality assurance, QA)*** *–* сукупність заходів, що охоплюють усі технологічні етапи розроблення, випуску та експлуатації ПЗ інформаційних систем на різних стадіях життєвого циклу ПЗ, для забезпечення його якості.

*При виконанні цих заходів для кожного продукту перевіряються****:***

* повнота та коректність документації;
* коректність процедур встановлення та запуску;
* ергономічність використання;
* повнота тестування.

У 80-х рр. ХХ ст. розмір проектів із створення програмних систем зріс настільки, що вручну стало неможливо тестувати ПЗ, тому активно стали розроблятися засоби автоматизації процесу тестування. Через десятиліття поняття якості ПЗ розширилося настільки, що було виділено окремий вид діяльності при створенні ПЗ – забезпечення якості (Quality assurance, QA). На цей час автоматизоване тестування значно поширене, а засоби автоматизованого тестування часто вбудовані у середовище програмування.

Для виконання заходів забезпечення якості ПЗ розробники часто використовують спеціальні групи контролю якості, які мають назву QA. Група QA всередині компанії фактично виконує роль вимогливого користувача. У деяких компаніях заборонено неформальне спілкування між групою розробників та групою тестувальників. Від відповідальності групи QA залежить успіх продукту. Якщо ПЗ не сподобалося з будь-яких причин користувачам, продукт назавжди втрачає репутацію і споживача.

Незнайдені на стадії розроблення помилки коштують дорого. Традиційна стратегія розроблення ПЗ підпорядковується фундаментальному правилу, що визначає, що впродовж роботи над проектом вартість внесення змін у створюване ПЗ збільшується за експонентою (рис. 2).

Фактично від того, наскільки якісно виконані початкові етапи розроблення ПЗ залежить його вартість. З метою підвищення якості розроблення та уникнення ризиків, пов’язаних із нечіткими або постійно змінюваними вимогами, була створена методологія XP (eXtreme Programming).

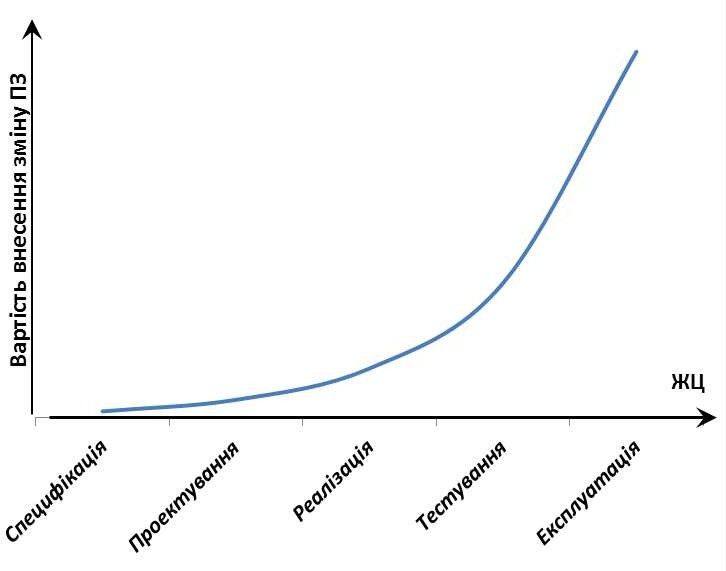


Рисунок 2 – Залежність вартості змін проектів від часу

**Тестування ПЗ**

*Тестування (Software Testing)* – діяльність, що виконується для оцінювання та поліпшення якості ПЗ. Ця діяльність базується на виявленні дефектів і проблем програмного забезпечення. Тестування ПЗ включає в себе діяльність із планування робіт (Test Management), проектування тестів (Test Design), виконнання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Програмування та тестування – різні за суттю види діяльності. Якщо програмування – процес синтезу, то тестування – процес аналізу.

Тестування потребує від виконавця в першу чергу уважності та педантичності. Тестувальник повинен шукати недоліки будь-де в системі.

Якщо початкове тестування для налагодження коду виконує сам програміст, то наступні етапи перевірки повинні готувати і виконувати інші особи, щоб перевірка була повноцінною, а не тільки для очікуваних проблемних місць.

Для планування потрібно мати уявлення про потрібні обсяги тестування. Важливі статистичні дані щодо тестування:

* на 1000 рядків коду програміст у середньому робить 100 помилок, 70% з яких усуваються на стадії налагодження коду;
* при системному тестуванні у відділі контролю якості на ліквідацію однієї помилки потрібно від чотирьох до шістнадцяти годин;
* для усунення помилки, що була виявлена у ході експлуатації, потрібно від 33 до 88 годин.

Ці дані показують, як важливо виявити та усунути проблеми ще на ранніх етапах розроблення. Цікавим підходом для підвищення якості програмування є парне програмування, що зменшує кількість помилок на 15 % порівняно із традиційним одиночним кодуванням.

Найдавнішим прийомом тестування є перевірка усіх вводів та виводів даних, передбачених та неприпустимих (complete testing). Програма повинна адекватно реагувати на помилкові ситуації, без втрати стійкості в роботі. Також при тестуванні потрібно перевірити виконання усіх передбачених функцій системи. Перелік таких тестів складається на ранніх етапах проектування паралельно із описом функцій програмного продукту.

**Життєвий цикл програмного забезпечення (SDLC – Software Development Life Cycle)** – період часу, який починається з моменту прийняття рішення про необхідність створення програмного продукту і закінчується в момент його повного вилучення з експлуатації. Цей цикл – процес побудови і розвитку програмного забезпечення.

**Етап 1 – Планування (Planning).** На цій фазі клієнт пояснює основні деталі і концепції проекту, оговорюється необхідний ресурс, час і бюджет, що необхідний для розробки.

**Етап 2 – Аналіз вимог (Requirements analysis).** Ця фаза розрахована для підготовки набору вимог. Потім йде етап узгодження вимог. Як результат ми маємо отримати узгоджений документ з вимогами.

**Етап 3 – Дизайн і розробка (Design & Development).** На цій фазі визначаються основні концепції дизайну програмного забезпечення. Після узгодження дизайну починається безпосередньо розроблення продукту.

**Етап 4 – Впровадження (Implementation).** Включає в себе програмування і отримання кінцевого продукту (бібліотеки, білди, документація).

**Етап 5 – Тестування (Testing).** На цій фазі проводиться перевірка на відповідність вимогам і підтвердження того, що продукт розроблений згідно з ними.

**Етап 6 – Оцінка (Evaluation).** На фазі оцінки (або пререлізу) продукт оцінюється замовником і вносяться останні уточнення.

**Етап 7 – Реліз (Release).** Заключна фаза розробки, враховуються уточнення, що зроблені замовником на фазі оцінки. Підготовка продукту в «коробці».

**Етап 8 – Підтримка (Support).** Фаза технічної підтримки продукту.

**Тестування ПЗ** (Software testing) – перевірка відповідності між реальною і очікуваною поведінкою програми. Тестування – це процес дослідження ПЗ з метою виявлення помилок і перевірки якості.

У більш широкому сенсі: Тестування – це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконання тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Тестування так само можна описати як процес верифікації та валідації того чи іншого програмного продукту, щоб дізнатися на скільки точно він задовольняє всім встановленим вимогам.

**Верифікація** (Verification – узгодження) – це процес оцінки системи або її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу (чи виконуються наші цілі, терміни, завдання, по розробці проекту, визначені на початку поточної фази.)

**Валідація** (Validation – затвердження) – це визначення відповідності ПЗ очікуванням і потребам користувача, вимогам до системи.

**Цілі і завдання процесу тестування**

Основною метою процесу тестування – є доказ того, що результат розробки відповідає пред’явленим до нього вимогам.

Основна завданням тестування ПЗ: отримання інформації про статус готовності заявленої функціональності системи або програми.

**Необхідні і достатні умови для проведення тестування**

Необхідними умовами істинності твердження А називаються умови, без дотримання яких А не може бути істинним. До них відносимо:

• Наявність об’єкта тестування, доступного для проведення випробувань.  
• Наявність виконавця (ів) (людина або машина, або комбінація людина + машина)

Достатніми називаються такі умови, за наявності (виконанні, дотриманні) яких твердження А є істинним. До них відносимо:

• Наявність об’єкта тестування, доступного для проведення випробувань

• Наявність виконавця (ів) (людина або машина, або комбінація людина + машина)

• Наявність плану тестування

• Наявність тест кейсів / тестів

• Наявність звіту, що підтверджує виконання завдань і досягнення цілей, з тестування об’єкта

**План Тестування** (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 дозволу .

**Тестовий випадок** (Test Case) – це артефакт, що описує сукупність кроків, конкретних умов і параметрів, необхідних для перевірки реалізації функції, що тестується або її частини.

Основні атрибути Test Case:

1) ID (номер),

2) Name (ім’я),

3) Preconditions (умови і параметри),

4) Steps (кроки до відтворення),

5) Expected result (очікуваний результат),

6) Actual result (фактичний результат),

7) Postconditions (постумови)

**Тест дизайн** (Test Design) – це етап процесу тестування ПЗ, на якому проектуються і створюються тестові випадки (тест кейси), відповідно з визначеними раніше критеріями якості і цілями тестування.

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

**Тестове Покриття** (Test Coverage) – це одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами.

**Специфікація Тест Кейсів** (Test Case Specification) – це рівень деталізації опису тестових кроків і необхідного результату, при якому забезпечується розумне співвідношення часу проходження до тестового покриття

**Час Проходження Тест Кейса** (Test Case Pass Time) – це час від початку проходження кроків тест кейса до отримання результату тесту.

**Особливості вимог програмного забезпечення.**

**Вимоги (Requirements)** до програмного забезпечення – сукупність тверджень щодо атрибутів, властивостей, або якостей програмної системи, що підлягає реалізації:

* **Одиничність** – Вимога описує одну і тільки одну річ.
* **Завершеність** – Вимога повністю визначена в одному місці і вся необхідна інформація присутня.
* **Послідовність** – Вимога не суперечить іншим вимогам і повністю відповідає зовнішній документації.
* **Атомарність** – Вимога не може бути розбита на ряд більш детальних вимог без втрати завершеності.
* **Відстежування** – Вимога повністю або частково відповідає діловим потребам як заявлено зацікавленими особами і задокументовано.
* **Актуальність** – Вимога не стала застарілою з часом.
* **Здійснимість** – Вимога може бути реалізовано в межах проекту.
* **Недвозначність** – Вимога коротко визначена без звернення до технічного жаргону та інших прихованих формулювань. Вона виражає об’єктивні факти, можлива одна і тільки одна інтерпретація. Визначення не містить нечітких фраз. Використання негативних тверджень заборонено.
* **Обов’язковість** – Вимога представляє певну характеристику, відсутність якої призведе до неповноцінності рішення, яка не може бути проігнорована.
* **Верифікованість** – Реалізованість вимоги може бути визначена через один з чотирьох можливих методів: огляд, демонстрація, тест чи аналіз.

**Методи тестування:**

1. Білий ящик (WhiteBox) – цей метод заснований на тому, що розробник тесту має доступ до коду програм і може писати код, який пов’язаний з бібліотеками ПЗ, що тестується.
2. Чорний ящик (Black Box) – цей метод заснований на тому, що тестувальник має доступ до ПЗ тільки через ті ж інтерфейси, що і замовник або користувач, або зовнішні інтерфейси, що дозволяють іншому комп’ютеру або іншому процесу підключитися до системи для тестування.
3. Сірий ящик (Grey Box) – поєднує елементи двох попередніх підходів.

**Фази тестування:**

1. Створення тестового набору (Test Suit) для конкретного середовища тестування (Testing Environment)
2. Прогон програми на тестах з отриманням протоколу результатів тестування (Test Log)
3. Оцінка результатів виконання програми на наборі тестів з метою прийняття рішення про продовження або зупинку тестування.

**Класи еквівалентності (Equivalence class):**

Підхід полягає в наступному: вхідні / вихідні дані розбиваються на класи еквівалентності, за принципом, що програма веде себе однаково з кожним представником окремого класу. Таким чином, немає необхідності тестувати всі можливі вхідні дані, необхідно перевірити по окремо взятому представнику класу.

Клас еквівалентності – це набір значень змінної, який вважається еквівалентним.

Тестові сценарії еквівалентні, якщо:

* Вони тестують одне і те ж;
* Якщо один з них знаходить помилку, то й інші виявлять її;
* Якщо один з них не знаходить помилку, то й інші не виявлять її.

**Еквівалентне розбиття:** Розробка тестів методом чорного ящика, в якому тестові сценарії створюються для перевірки елементів еквівалентної області. Як правило, тестові сценарії розробляються для покриття кожній області як мінімум один раз.

[](https://qalearning.com.ua/wp-content/uploads/2015/01/2.png)

Приклад: Припустимо, ми тестуємо Інтернет-магазин, який продає олівці. У замовленні необхідно вказати кількість олівців (максимум для замовлення – 1000 штук). Залежно від замовленої кількості олівців змінюється вартість:

1 – 100 – 10 грн. за олівець;

101 – 200 – 9 грн. за олівець;

201 – 300 – 8 грн. за олівець і т.д.

З кожною новою сотнею, ціна зменшується на гривню.

Якщо тестувати «в лоб», то, щоб перевірити всі можливі варіанти обробки замовленої кількості олівців, потрібно написати дуже багато тестів (згадуємо, що можна замовити аж 1000 штук), а потім ще все це і протестувати. Спробуємо застосувати розбиття на класи еквівалентності. Очевидно, що наші вхідні дані можна розділити на наступні класи еквівалентності:

Невалідне значення:> 1000 штук;

Невалідне значення: <= 0;

Валідне значення: від 1 до 100;

Валідне значення: від 101 до 200;

Валідне значення: від 201 до 300;

Валідне значення: від 301 до 400;

Валідне значення: від 401 до 500;

Валідне значення: від 501 до 600;

Валідне значення: від 601 до 700;

Валідне значення: від 701 до 800;

Валідне значення: від 801 до 900;

Валідне значення: від 901 до 1000.

На основі цих класів ми і складемо тестові сценарії. Отже, якщо взяти по одному представнику з кожного класу, то отримуємо 12 тестів.

**Рівні тестування**

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 і т.д. 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. Оптимально.
* 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 ПЗ.**

Всі види тестування програмного забезпечення, залежно від переслідуваних цілей, можна умовно розділити на наступні групи:

* Функціональні (Functional testing)
* Нефункціональні (Non-functional testing)
* Пов’язані зі змінами (Regression testing)

**Функціональні види тестування**

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing ).

Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

Функціональне тестування (Functional testing)

Тестування безпеки (Security and Access Control Testing)

Тестування взаємодії (Interoperability Testing)

**Нефункціональні види тестування**

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, “Як” система працює. Далі перераховані основні види нефункціональних тестів:

Всі види тестування продуктивності:

тестування навантаження (Performance and Load Testing)

стресове тестування (Stress Testing)

тестування стабільності або надійності (Stability / Reliability Testing)

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)

**Тестування, пов’язане зі змінами**

Після проведення необхідних змін, таких як виправлення бага / дефекту, програмне забезпечення повинне бути перетестоване для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності здійсненого виправлення дефекту:

Димове тестування (Smoke Testing)

Регресійне тестування (Regression Testing)

Тестування збірки (Build Verification Test)

Санітарне тестування або перевірка узгодженості / справності (Sanity Testing)

**Функціональне тестування** (Functional testing)

Тестування функціональності може проводиться у двох аспектах:

* вимоги
* бізнес-процеси

Тестування в аспекті «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріоритезувати вимоги на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього пріоритезувати тестові сценарії (test cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал.

Тестування в сенсі «бізнес-процеси» використовує знання цих самих бізнес-процесів, які описують сценарії щоденного використання системи. У цьому випадку тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

Переваги функціонального тестування:

імітує фактичне використання системи;

Недоліки функціонального тестування:

можливість упущення логічних помилок у програмному забезпеченні;

ймовірність надмірного тестування.

Досить поширеною є автоматизація функціонального тестування.

**Тестування безпеки (Security and Access Control Testing)** – це стратегія тестування, що використовується для перевірки безпеки системи, а також для аналізу ризиків, пов’язаних із забезпеченням цілісного підходу до захисту додатків, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Загальна стратегія безпеки ґрунтується на трьох основних принципах: конфіденційність, цілісність, доступність.

**Конфіденційність** – це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов користувач авторизований отримати доступ до даного ресурсу.

**Цілісність** – Існує два основних критерії при визначенні поняття цілісності:

**Довіра** – Очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів.

**Пошкодження і відновлення** – у разі коли дані пошкоджуються або неправильно змінюються авторизованим або авторизованим користувачем, потрібно визначити наскільки важливою є процедура відновлення даних.

**Доступність** являє собою вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об’єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності.

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

**Види тестів**

Одним з найбільш об'єктивних методів оцінки якості програм є її випробування. *Випробування програми* може проводитися, наприклад, з метою визначення міри відповідності готової програми вимогам, що сформульовані у технічному завданні.

Так само шляхом випробування можуть бути отримані точні оцінки таких параметрів як:

* середній час вирішення завдання;
* максимальний обсяг необхідний оперативної пам'яті;
* показники завантаження зовнішніх пристроїв необхідних для оцінки вартості рішення задачі.

Іншою, не менш важливою, метою проведення випробувань є спрямований пошук помилок. З якою б метою не проводилося випробування програми, одним з найважливіших питань є підбір вхідних даних програми. Такі, спеціально підібрані з певними цілями, вихідні дані називаються тестами**.** Випробування програми тестами – тестування – починається вже в процесі її налагодження.

Перший тестовий приклад пропонується програмою відразу після її трансляції. Далі тестування і налагодження тісно переплітаються між собою. Якщо черговий тест не проходить, тобто програма видає результат, що не відповідає тестовим даним, то виникає проблема знайти і виправити помилку, а це вже налагодження**.**

З іншого боку, для локалізації помилки необхідно вибрати такі вихідні дані, тобто тестові приклади, які б сприяли найбільш рельєфному прояву виявленої, але ще не локалізованої помилки.

*Тестування* – етап процесу створення програми, що полягає в підборі тестових прикладів і випробування на них програми, з метою виявлення в ній помилки.

*Налагодження* – етап процесу створення програми, що полягає в локалізації та виправленні помилок, виявлених у ході трансляції та тестування.

*Етап тестування включає три основних елементи:*

1. генерація тестових прикладів, контроль процесу тестування;
2. аналіз вихідних і проміжних результатів програми;
3. врахування внеску кожного тестового прикладу в процес тестування.

Для першого і другого елементів вихідними даними є документація програми, на основі якої генеруються тестові приклади, що забезпечують режим виконання програми, визначає ступінь відповідності отриманого результату зазначеного в прикладі.

Третій елемент визначається критерієм повноти тестування, який тісно пов'язаний з конкретним способом тестування.

Тестування ведеться на чотирьох рівнях, які розглядали вище. Цілком природно тестування ув'язується з організацією розробки програм. Тому, два способи утворюють базис, на якому будуються різні методи тестування.

*Висхідне тестування* передбачає, що програма збирається і тестується знизу вгору. Модулі (компоненти самого нижнього рівня) тестуються автономно. Надійність тестування цих модулів визначає успіх подальшого процесу. Далі відбувається перехід до модулів, які звертаються до вже відтестованих модулів. На цьому етапі виникає необхідність перевірки інтерфейсів.

Для реалізації висхідного тестування необхідно для кожного модуля написати невелику керуючу підпрограму - драйвер. У розпорядження драйвера надаються значення вхідних змінних і структури даних. Драйвер послідовно викликає модуль, що тестується, при кожному виклику пропонуючи йому новий тестовий приклад. Основні недоліки:

* + серйозні помилки в специфікаціях, алгоритмах і інтерфейсах можуть проявлятися лише при тестуванні комплексів вищого рівня, тобто на завершальній стадії.
  + Необхідність розробки драйверів і тестів для кожного рівня тестування, що веде до великого обсягу додаткових програм, які стають даремними при завершенні роботи над комплексом.

*Спадне проектування****.*** Автономно тестується тільки головна програма. По завершенні тестування головного комплексу до нього послідовно приєднуються комплекси та компоненти наступного рівня і т.д., до тих пір, поки не буде зібрана і випробувана вся програма.

Як же тестувати комплекси, в той час як компоненти, що входять в них, ще не перевірені і можливо ще і не написані? Для імітації функцій ще не створених модулів використовуються заглушки, які імітують роботу відсутнього модуля.

Недоліки спадного тестування збігаються з вадами спадного проектування. Підвищуються вимоги до складності і якості заглушок, які потім ліквідуються. Переваги:

* + Метод дозволяє поєднати тестування модуля, тестування сполучень і тестування вхідних функцій.
  + Рівномірний розподіл роботи з тестування протягом усього періоду створення комплексу - це дозволяє виявити помилки в головному модулі на ранній стадії розробки.

На практиці рідко вдається використовувати один спосіб, існує ряд комбінованих способів.

*Методи тестування компонентів*

Перший метод розглядає тестування програм як «чорний ящик», при цьому внутрішня структура програми (компонента) не враховується, тести будуються на підставі функціональних властивостей програми, тобто спираючись на її функціональні специфікації. Такий підхід називається *функціональним тестуванням****.***

*Структурне тестування****,*** при його використанні враховується

внутрішня структура програми. Аналіз проходження даних від входу до виходу, ця інформація використовується для раціональної організації тестування.

*Для оцінки повноти тестування використовують три критерії:*

1. тестування вважається закінченим, якщо кожен оператор був виконаний хоча б раз;
2. тестування вважається закінченим, якщо в процесі вирішення тестових прикладів по кожній дузі блок-схеми програми був здійснений хоча б один перехід;
3. тестування закінчується, якщо в процесі вирішення тестового прикладу кожен шлях від входу до виходу пройдений хоча б раз.

*Налагодження -* процес пошуку та усунення помилок у програмі, що спирається на результати самої програми, повідомлення транслятора, і оперативної системи. Важливою особливість процесу налагодження є можливість робити експерименти на ЕОМ з метою виявлення помилок. Але експеримент повинен вестися в суворій відповідності з планом.

Процес налагодження полягає в багаторазовому повторенні трьох етапів:

* 1. виявлення помилки;
  2. локалізація помилки;
  3. виправлення помилки.

*Виявлення помилки* при налагодженні здійснюється шляхом прорахунку на ЕОМ спеціально підібраних завдань, результати вирішення яких заздалегідь відомі. Якщо контрольний приклад вирішено правильно, то підбирається більш складне завдання. Якщо програма не йде, то в ній є, принаймні, одна помилка.

Наступний етап полягає у встановленні точного місця знаходження помилки. *Локалізація помилок* є процес вирішення неправильних завдань.

Етап *виправлення помилки* є найбільш простим, але основна складність не внести нову помилку при виправленні.