**Лекція 16. Тест-дизайн. Тест-кейси**

**Підготовка тестування програмного продукту**

Одним з інструментів вимірювання якості розроблюваного програмного забезпечення (ПЗ) є тестування, який по суті є процесом технічного дослідження на вимогу замовників для вияву інформації про якість продукту відносно контексту, в якому він має використовуватись. До цього процесу входить виконання програми з метою знайдення помилок, що було розглянуто на попередніх заняттях. Щодо інших аспектів оцінювання якості ПЗ таких як надійність, супроводжуваність, практичність, ефективність, мобільність, функціональність за стандартом ISO 9126, а також заданих замовником, то не їх завжди можна оцінити через метрики. Що дає можливість визначити, що якість - це суб'єктивне поняття. Тому тестування не може повністю забезпечити коректність ПЗ з точки зору заданих визначень якості. Воно тільки порівнює стан і поведінку продукту зі специфікацією. При цьому треба розрізняти тестування програмного забезпечення і забезпечення якості програмного забезпечення, до якого належать усі складові ділового процесу, а не тільки тестування.

Тестування виконується протягом всього життєвого циклу ПЗ, починаючи від визначення вимог і закінчуючи етапом експлуатації, на кожному перевіряючи відповідність програм визначеним вимогам. Як ітераційний процес тестування не може вважатися остаточно завершеним або, як висловився професор Едсжер Дейкстра у 1972 році: "Тестування програм може служити доказом наявності помилок, але ніколи не доведе їхню відсутність!" Таким чином, при підготовці тестування стоять два завдання.

*1. П*ідготувати такий набір тестів і застосувати до них ПЗ, щоб виявити в нім по можливості більше число помилок. Проте чим довше продовжується процес тестування (і відладки в цілому), тим більшою стає вартість ПЗ.

2. *В*изначити момент закінчення відладки ПЗ (або окремою його компоненти). Ознакою можливості закінчення відладки є повнота обхвату пропущеними через ПЗ тестами (тобто тестами, до яких застосоване ПЗ) безлічі різних ситуацій, що виникають при виконанні програм ПЗ, і відносний рідкісний прояв помилок в ПЗ на останньому відрізку процесу тестування. Останнє визначається відповідно до необхідного ступеня надійності ПЗ, вказаної в специфікації його якості.

Для оптимізації набору тестів, тобто для підготовки такого набору тестів, який дозволяв би при заданому їх числі (або при заданому інтервалі часі, відведеному на тестування) виявляти більше число помилок, необхідно, по-перше, заздалегідь планувати цей набір і, по-друге, використовувати раціональну стратегію планування (проектування ) тестів.

Проектування тестів бажано починати відразу ж після завершення етапу зовнішнього опису ПЗ. Можливі різні підходи до вироблення стратегії проектування тестів, які можна умовно графічно розмістити (див. рис. 1) між наступними двома крайніми підходами. Лівий крайній підхід (тестування по відношенню до специфікації) полягає в тому, що тести проектуються тільки на підставі вивчення специфікацій ПЗ (зовнішнього опису, опису архітектури і специфікації модулів). Будова модулів при цьому ніяк не враховується, тобто вони розглядаються як чорні ящики. Фактично такий підхід вимагає повного перебору всіх наборів вхідних даних, оскільки при використанні як тести тільки частині цих наборів деякі ділянки програм ПЗ можуть не працювати ні на якому тесті і, значить, помилки, що містяться в них, не виявлятимуться. Проте тестування ПЗ повним безліччю наборів вхідних даних практично нездійсненно. Правий крайній підхід (тестування по відношенню до текстів програм ) полягає в тому, що тести проектуються на підставі вивчення текстів програм з метою протестувати всі шляхи виконання кожній програм ПЗ. Якщо взяти до уваги наявність в програмах циклів із змінним числом повторень, то різних шляхів виконання програм ПЗ може опинитися також надзвичайно багато, так що їх тестування також буде практично нездійсненно.



Рис. 1. Різні підходи до проектування тестів.

Оптимальна стратегія проектування тестів розташована усередині інтервалу між цими крайніми підходами, але ближче до лівого краю. Вона включає проектування значної частини тестів по специфікаціях, виходячи з принципів: на кожну використовувану функцію або можливість - хоч би один тест, на кожну область і на кожну межу зміни якої-небудь вхідної величини - хоч би один тест, на кожен особливий випадок або на кожну виняткову ситуацію, вказані в специфікаціях, - хоч би один тест. Але вона вимагає також проектування деяких тестів і по текстах програм, виходячи з принципу (як мінімум): кожна команда кожної програми ПЗ повинна пропрацювати хоч би на одному тесті.

Оптимальну стратегію проектування тестів можна конкретизувати за принципом: для кожного програмного документа (включаючи тексти програм), що входить до складу ПЗ, повинні проектуватися свої тести з метою виявлення в ньому помилок. Майерс навіть визначає різні види тестування залежно від вигляду програмного документа, на підставі якого будуються тести. До таких документів відносяться визначення вимог до ПЗ, специфікація якості ПЗ, функціональна специфікація ПЗ, опис архітектури ПЗ і тексти програм ПЗ [7].

Згідно Майерсу [7] тестування повинне бути організоване за принципами:

1) кожен тест повинен включати опис очікуваних результатів роботи програми, щоб можна було швидко з'ясувати наявність або відсутність помилок в ній;

2) бажано, щоб окрім автора програми тестування проводили і інші спеціалісти, оскільки виявлення недоліків в своїй діяльності протирічить людській психології. Однак, відладка програми найефективніше виконується саме автором програми;

3) організація-розробник ПЗ не повинна "одноосібно" тестувати програмний продукт, потрібно організовувати бета-тестування з залученням інших організацій;

4) результати кожного тесту повинні бути документовані і детально вивчені, щоб не пропустити малопомітну на перший погляд помилку в програмі;

5) тести для неправильних (непередбачуваних) даних повинні підбиратися так само ретельно, як і для правильних (передбачуваних) вхідних даних;

6) при аналізі результатів кожного тесту необхідно перевіряти, чи не робить програма того, що вона не повинна робити;

7) потрібно зберігати використані тести (для підвищення ефективності повторного тестування програми після її модифікації або установки у замовника);

8) тестування не повинно плануватися виходячи з припущення, що в програмі не будуть виявлені помилки (зокрема, потрібно виділяти на тестування достатньо часових і матеріальних ресурсів);

9) потрібно враховувати так званий "принцип накопичення помилок": вірогідність наявності не виявлених помилок в деякій частині програми прямо пропорційна числу помилок, вже виявлених в цій частині;

10) потрібно завжди пам'ятати, що тестування - творчий процес, і не відноситися до нього як до рутинного заняття.

Процес тестування об'єднує різні способи тестування в сплановану послідовність кроків, які приводять до успішної побудови програмної системи (ПС). Методика тестування ПС може бути представлена у вигляді спіралі, що розгортається (рис. 2).

На початку здійснюється *тестування елементів (модулів), яке* перевіряє результати етапу *кодування* ПС. На другому кроці виконується *тестування інтеграції,* орієнтоване на виявлення помилок етапу *проектування* ПС. На третьому обороті спіралі проводиться *тестування*



Рис.2. Спіраль процесу тестування ПС

Кожен крок процесу тестування характеризується таким чином.

1. *Тестування елементів.* Мета — індивідуальна перевірка кожного модуля. Використовуються способи тестування «білого ящика».

2. *Тестування інтеграції.* Мета — тестування збірки модулів в програмну систему. В основному застосовують способи тестування «чорного ящика».

3. *Тестування правильності.* Мета — перевірити реалізацію в програмній системі всіх функціональних і поведінкових вимог, а також вимоги ефективності. Використовуються виключно способи тестування «чорного ящика».

4. *Системне тестування.* Мета — перевірка правильності об'єднання і взаємодії всіх елементів комп'ютерної системи, реалізації всіх системних функцій.

Організація процесу тестування у вигляді еволюційної спіралі, що розгортається, забезпечує максимальну ефективність пошуку помилок. Проте виникає питання — коли закінчувати тестування? На практицізазвичай застосовують статистичний критерій: «Можна з 95%-ю впевненістю сказати, що проведено достатнє тестування, якщо вірогідність безвідмовної роботи ПП протягом 1000 годин складає щонайменше 0,995».

На кожному кроці тестування застосовуються різні види тестування, залежно від типу ПС та її обсягів.

**Умови тестування**

Тестування ПЗ – це процес дослідження ПЗ з метою отримання інформації про якість програмного продукту, а саме відповідність специфікації, технічному завданню, або вимогам замовника ПЗ. Практичний підхід до тестування ПЗ особливу увагу приділяє процесам тестування на фоні стрімкого прискорення процесу розробки ПЗ. Цей підхід орієнтований на використання спеціалістами з тестування ПЗ тестових робіт. Швидкість і ефективність розробки ПЗ залежить від того наскільки процес тестування вписується в загальний життєвий цикл розробки ПЗ і від ефективності використання технології тестування. Тестування - це одна з технік контролю якості, що включає в себе діяльність з планування робіт (Test Management), проектуванню тестів (Test Design), виконанню тестування (Test Execution) і аналізу отриманих результатів (Test Analysis). ***Необхідними умовами*** для тестування є наявність :

• об'єкта тестування, доступного для проведення іспитів;

• виконавця(ів) (залежно від виду проведених іспитів їм може бути як людина, так і машина або комбінація людина + машина).

***Достатніми умовами*** для тестування є наявність:

• об'єкта тестування, доступного для проведення іспитів;

• виконавця(ів) (залежно від виду діяльності на різних фазах їм може бути як людина, так і машина або комбінація людина + машина);

• плану тестування;

• тест кейсів / тестів;

• звіту, що підтверджує виконання задач і досягнення цілей, по тестуванню об'єкта.

Вимоги до документів з тестування надаються згідно відповідних стандартів, перелік документів такий:

• тест план (тест план IEEE 829, тест план RUP, план приймально – здавальних випробувань RUP, план проведення навантажувального тестування);

• тест дизайн специфікації (тест дизайн специфікація MSF, тест дизайн специфікація IEEE 829-1998);

• тестовий випадок (test case);

• звіт про помилку (bug report).

**Сутність тестування та документування процесу**

*Тестування програмного забезпечення (Software Testing)* – це перевірка відповідності між реальною і очікуваною поведінкою програми, що здійснюється на кінцевому наборі тестів, обраних певним чином. (IEEE Guide to Software Engineering Body of Knowledge, SWEBOK, 2004)

У більш широкому змісті, тестування - це одна з технік контролю якості, що включає в себе діяльність з планування робіт (Test Management), проектуванню тестів (Test Design), виконанню тестування (Test Execution) і аналізу отриманих результатів (Test Analysis).

*Верифікація (verification)* програми і її компонентів з метою визначення чи задовольняють результати поточного етапу розробки умовам, сформованим на початку цього етапу (IEEE). Тобто чи виконуються цілі, терміни, задачі з розробки проекту, визначені на початку поточної фази.

*Валідація (validation)* - це визначення відповідності розроблювального ПЗ очікуванням і потребам користувача, вимогам до системи (BS7925-1).

***План тестування*** *(Test Plan)* - це документ, що описує обсяг робіт з по тестування, починаючи з опису об'єкта, стратегії, розкладу, критеріїв початку і закінчення тестування, до необхідного в процесі роботи устаткування, спеціальних знань, а також оцінки ризиків з варіантами їхнього рішення.

Розглянемо визначення.

*Тест дизайн (Test Design)* - це етап процесу тестування ПЗ, на якому проектуються і створюються тестові випадки (тест кейси), відповідно до визначених раніше критерій якості і цілей тестування.

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

*Звіт про помилку/Дефект Репорт (Bug Report)* - це документ, що описує ситуацію або послідовність дій, що призвели до некоректної роботи об'єкта тестування, із вказівкою причин і очікуваного результату.

*Тестове Покриття (Test Coverage)* - це одна з метрик оцінки якості тестування, що представляє із себе щільність покриття тестами вимог або коду, що виконується.

*Деталізація Тест Кейсів (Test Case Detalization)* - 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 одержання результату тесту.

Залежно від переслідуваних цілей види тестування можна умовно розділити на наступні типи: функціональне, не функціональне, пов'язані зі змінами.

Таблиця 1 – Класифікація видів тестування ПЗ

| А Функціональне | Б Нефункціональне | В Пов'язані зі змінами |
| --- | --- | --- |
| 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 зовнішнє поводження системи. Далі перераховані одні з найпоширеніших видів функціональних тестів | Нефункціональне тестування описує тести, необхідні для визначення характеристик ПЗ, що можуть бути вимірювані різними величинами. У цілому, це тестування того, "Як" система працює. Основні види нефункціональних тестів: | Після проведення необхідних змін, таких як виправлення помилки - бага/дефекту, ПЗ має бути знов протестоване для підтвердження того факту, що проблема була дійсно вирішена. перераховані види тестування, які необхідно проводити після установки ПЗ, для підтвердження працездатності додатка або вірності здійсненого виправлення дефекту: |
| А.1Функціональне тестування (Functional testing) | Б.1 Види тестування продуктивності: Б.1.1 Навантажувальне тестування (Performance and Load Testing) Б.1.2 Стресове тестування (Stress Testing) Б.1.3 Тестування стабільності або надійності (Stability / Reliability Testing)  Б.1.4 Об'ємне тестування (Volume Testing) | В.1 Димове тестування (Smoke Testing) |
| А.2 Тестування безпеки (Security and Access Control Testing) | Б.2. Тестування установки (Installation testing | В.2 Регресійне тестування (Regression Testing) |
| А.3 Тестування взаємодії (Interoperability Testing):  А.3.1 Тестування сумісності  (compatibility testing); А.3.2 Інтеграційне тестування (integration testing). | Б.3 Тестування зручності користування (Usability Testing) | В.3 Тестування зборки (Build Verification Test |
|  | Б.4 Тестування на відмовлення і відновлення (Failover and Recovery Testing) | В.4 Санітарне тестування -перевірка погодженості/ справності (Sanity Testing) |
|  | Б.5 Конфігураційне тестування (Configuration Testing) |  |

*А.1 Функціональне тестування (Functional Testing)* розглядає заздалегідь зазначену поведінку і ґрунтується на аналізі специфікацій функціональності компонента або системи в цілому. Функціональні тести ґрунтуються на функціях, виконуваних системою, і можуть проводитися на всіх рівнях тестування (компонентному, інтеграційному, системному, приймальному). Як правило, ці функції описуються у вимогах, функціональних специфікаціях або у виді випадків використання системи (use cases).

Тестування функціональності може проводиться в двох аспектах:

• вимоги;

• бізнеси-процеси.

*Тестування в перспективі «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків* (Test Cases). У цьому випадку необхідно зробити перелік того, що буде тестуватися, а що ні, визначають пріоритетність вимог на основі ризиків (якщо це не зроблено в документі з вимогами), а на основі цього переліку пріоритетів формуються тестові сценарії (test cases). Це дозволяє сфокусуватися при тестуванні на важливішому функціоналі. Тестування в перспективі «бізнеси-процеси» використовує знання цих самих бізнесів-процесів, що описують сценарії щоденного використання системи. У цій перспективі тестові сценарії (test scripts), як правило, ґрунтуються на випадках використання системи (use cases).

*А.2 Тестування безпеки (Security and Access Control Testing)* – це перевірка безпеки системи, а також аналіз ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатка, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних. Тестування безпеки може виконуватися як автоматизовано так і в ручну, включаючи перевірку як позитивних, так і негативних тестових випадків. Ґрунтується на трьох основних принципах - *це конфіденційність, цілісність і доступність* (confidentiality, integrity, availability)

*А.3 Тестування взаємодії (Interoperability Testing)* – це функціональне тестування, що перевіряє здатність додатка взаємодіяти з одним і більш компонентами або системами, що включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).

*Б.1 Види тестування продуктивності (Performance testing)* Задачею *тестування продуктивності (Performance testing)* є визначення масштабованості додатка під навантаженням, при цьому відбувається:

• вимір часу виконання обраних операцій при визначених інтенсивностях виконання цих операцій;

• визначення кількості користувачів, що одночасно працюють з додатком;

• визначення границь прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій);

• дослідження продуктивності на високих, граничних, стресових навантаженнях.

*Б.1.1 Навантажувальне тестування (Load vs Performance Testing).*

В англомовній термінології ви можете так само знайти ще один вид тестування - Load Testing - тестування реакції системи на зміну навантаження (у межах припустимого). Load і Performance переслідують одну й теж саму мету: перевірка продуктивності (часів відгуку) на різних навантаженнях. Власне тому їх не розділяють.

*Б.1.2 Стресове тестування (Stress Testing)* дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом у даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійна зміна конфігурації сервера. Також однієї з задач при стресовому тестуванні може бути оцінка деградації продуктивності, у такий спосіб мети стресового тестування можуть перетинатися з цілями тестування продуктивності.

*Б.1.3 Тестування стабільності або надійності (Stability / Reliability Testing)* – це перевірка працездатності додатка при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження. Часи виконання операцій можуть грати в даному виді тестування другорядну роль. При цьому на перше місце виходить відсутність витоків пам'яті, перезапусків серверів під навантаженням і інші аспекти, що впливають саме на стабільність роботи.

*Б.1.4 Об'ємне тестування (Volume Testing)* -це одержання оцінки продуктивності при збільшенні обсягів даних у базі дані додатки, при цьому відбувається:

• вимір часу виконання обраних операцій при визначених інтенсивностях виконання цих операцій;

• може вироблятися визначення кількості користувачів, що одночасно працюють з додатком

*Б.2 Тестування установки (Installation Testing)* направлено на перевірку успішної інсталяції і настроювання, а також відновлення або видалення ПЗ. Інсталяція відбувається автоматично вручну та за допомогою візардів.

*Б.3 Тестування зручності користування (Usability Testing)* - це метод тестування, спрямований на встановлення ступеня зручності використання, навчання, зрозумілості і привабливості для користувачів розроблювального продукту в контексті заданих умов.

*В. Тестування, що пов'язане зі змінами*

*В.1 Димове тестування (Smoke Testing)* спрямовано на поверхневу перевірку всіх модулів додатка на предмет працездатності і наявність швидкого знаходження критичних і дефектів, що блокують. За результатами димового тестування робиться висновок про те, чи приймається чи ні установлена версія ПЗ в тестування, експлуатацію або на постачання замовникові. Для полегшення роботи, економії часу і людських ресурсів рекомендується автоматизувати димові тести.

*В.2 Регресійне тестування (Regression Testing)* спрямовано на перевірку змін, зроблених у додатку або навколишнім середовищі (налагодження дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер додатка), для підтвердження того факту, що існуюча раніше функціональність працює як і колись. Регресійними можуть бути тести як функціональні, так і не функціональні. Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки і тестування. Це дає гарантію того, що зміни в новій версії додатка не зашкодили вже існуючу функціональність. Рекомендується робити автоматизацію регресійних тестів, для прискорення наступного процесу тестування і виявлення дефектів на ранніх стадіях розробки програмного забезпечення. Сам по собі термін "Регресійне тестування", залежно від контексту використання може мати різний сенс.

3 основних типи регресійного тестування:

• регресія помилок - багів (Bug regression) - спроба довести, що виправлена помилка насправді не виправлена

• регресія старих помилок - багів (Old bugs regression) - спроба довести, що недавня зміна коду або даних зламало виправлення старих помилок, тобто старі помилки - баги стали знову відтворюватися.

• регресія побічного ефекту (Side effect regression) - спроба довести, що недавня зміна коду або даних зламало інші частини розроблювального додатка

*В.3 Тестування зборки (Build Verification Test)* Тестування спрямоване на визначення відповідності, випущеної версії, критеріям якості для початку тестування. По своїм цілям є аналогом Димового Тестування, спрямованого на приймання нової версії в подальше тестування або експлуатацію. Вглибину воно може проникати далі, залежно від вимог до якості випущеної версії.

*В.4 Санітарне тестування або перевірка погодженості/справності (Sanity Testing).* Вузькоспеціалізоване тестування достатнє для доказу того, що конкретна функція працює згідно заявленим у специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності визначеної частини додатка після змін зроблених у ньому або навколишньому середовищу. Звичайно виконується вручну.

**Зміст тестового плану**

Вступ (Introduction) подається в довільній формі

Мета (Purpose)

• Виявлення існуючих проектів і програмних компонентів, які необхідно перевірити.

• Перелік вимог для проведення випробування.

• Рекомендації щодо опису стратегії тестування.

• Визначення необхідних ресурсів і забезпечення оцінки випробувань.

• Перелік тестових елементів проекту.

Довідкова інформація (Background**)**

Опис елементів тестування (компоненти, додатки, системи тощо).

Інформація щодо основних функцій і можливостей, архітектури, короткої історії проекту

Галузь застосування (Scope), описують:

• Типи, наприклад, групи, інтеграцію, систему, етапи тестування (функціональність і п

• Перелік особливостей функцій, що будуть протестовані.

• Перелік пропозицій, що можуть вплинути на проектування, розробку тестування.

• Перелік усіх ризиків й непередбачуваних обставин, що можуть вплинути на розробку

• Перелік обмежень, які можуть вплинути на проектування, розробку тестування.

Визначення проекту (Project Identification)

**Розробка тестових випадків (test case)**

Тестування на різних рівнях виробляється протягом усього життєвого циклу розробки і супроводу ПЗ. Рівень тестування визначає те, над чим виробляються тести: над окремим модулем, групою модулів або системою, у цілому. Проведення тестування на всіх рівнях системи - це основа успішної реалізації і здачі проекту. Рівні тестування:

• компонентне або модульне тестування (Component Testing or Unit Testing)

• інтеграційне тестування (Integration Testing)

• системне тестування (System Testing)

• приймальне тестування (Acceptance Testing)

*Компонентне (модульне) тестування (Component or Unit Testing)* перевіряє функціональність і шукає дефекти в частинах додатка, які доступні і можуть бути протестовані окремо (модулі програм, об'єкти, класи, функції тощо).

Звичайно компонентне (модульне) тестування проводиться викликаючи код, який необхідно перевірити і за підтримкою середовищ розробки, таких як фреймворки (frameworks - каркаси) для модульного тестування або інструменти для налагодження.

Усі знайдені дефекти, як правило виправляються в коді без формального їхнього опису в системі менеджменту помилок/дефектів - багів (Bug Tracking System). Один з найбільш ефективних підходів до компонентного (модульного) тестування - це підготовка автоматизованих тестів до початку основного кодування (розробки) програмного забезпечення.

Це називається «розробка від тестування» (test-driven development) або «підхід тестування спочатку» (test first approach). При цьому підході створюються й інтегруються невеликі частини коду, напроти яких запускаються тести, написані до початку кодування. Розробка ведеться доти поки всі тести не будуть успішними. *Інтеграційне тестування (Integration Testing)* призначене для перевірки зв'язку між компонентами, а також взаємодії з різними частинами системи (операційною системою, устаткуванням або зв'язком між різними системами).

Рівні інтеграційного тестування:

* *компонентний інтеграційний рівень* (Component Integration testing). Перевіряється взаємодія між компонентами системи після проведення компонентного тестування;
* *системний інтеграційний рівень* (System Integration Testing).

Перевіряється взаємодія між різними системами після проведення системного тестування.

Підходи до інтеграційного тестування:

• *знизу нагору* (Bottom Up Integration). Усі низькорівневі модулі, процедури або функції збираються разом і потім тестуються. Після чого збирається наступний рівень модулів для проведення інтеграційного тестування. Даний підхід вважається корисним, якщо всі або практично всі модулі, рівня, що розробляється готові. Також даний підхід допомагає визначити за результатами тестування рівень готовності додатка;

• *зверху вниз* (Top Down Integration). Спочатку тестуються усі високорівневі модулі, і поступово один за іншим додаються низькорівневі. Усі модулі більш низького рівня симулюються заглушками з аналогічною функціональністю, потім в міру готовності вони заміняються реальними активними компонентами;

• *великий вибух* ("Big Bang" Integration). Всі або практично всі розроблені модулі збираються разом у вигляді закінченої системи або її основної частини, і потім проводиться інтеграційне тестування. Такий підхід дуже гарний для збереження часу. Однак якщо тест кейси і їхні результати записані не вірно, то сам процес інтеграції сильно ускладниться, що стане перешкодою для команди тестування при досягненні основної мети інтеграційного тестування.

Основною задачею *системного тестування* 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 користувальницького рівня, несумісність з оточенням, непередбачені сценарії використання, відсутня або невірна функціональність, незручність використання тощо.

Для мінімізації ризиків, пов'язаних з особливостями поведінки в системі в будь-якому середовищі, під час тестування рекомендується використовувати оточення максимальне наближене до того, на яке буде встановлений продукт після видачі. Можна виділити два підходи до системного тестування:

* на базі вимог (requirements based).Для кожної вимоги пишуться тестові випадки (test cases), що перевіряють виконання даної вимоги;
* на базі випадків використання (use case based). На основі представлення про способи використання продукту створюються випадки використання системи (Use Cases).

По конкретному випадку використання можна визначити один або більш сценаріїв. На перевірку кожного сценарію пишуться тест кейси (test cases), які мають бути протестовані.

Формальний процес *приймального тестування*, що перевіряє відповідність системи вимогам і проводиться з метою:

• визначення чи задовольняє система приймальним критеріям;

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

Приймальне тестування виконується на підставі набору типових тестових випадків і сценаріїв, розроблених на підставі вимог до даного додатка. Рішення про проведення приймального тестування приймається, коли:

• продукт досяг необхідного рівня якості;

• замовник ознайомлений із планом приймальних робіт (Product Acceptance Plan) або іншим документом, де описаний набір дій, пов'язаних із проведенням приймального тестування, дата проведення, відповідальні особи тощо.

Фаза приймального тестування триває доти, поки замовник не виносить рішення про відправлення додатка на доробку або видачі додатка. *Тестовий випадок (test case)* ***-*** сукупність вхідних даних тесту, умови виконання і очікуваних результатів, які розроблені для конкретної мети. Тестовий випадок - це найменша одиниця тестування, яку можна самостійно виконати від початку до кінця.

Розглянемо особливості заповнення полів шаблону тестування. *Ідентифікатор тестового випадку -* включає номер версії тесту.

*Власник тесту* – ПІБ особи, що експлуатує тест (воно може не співпадати з ПІБ автора тесту).

*Дата останнього перегляду* – ця інформація визначає актуальність тесту.

*Назва тесту* - опис назви тесту, що дозволяє його легко знайти і зрозуміти його призначення. Не рекомендується вживати назви, що не несуть ніякого сенсового навантаження, наприклад, "xxxLLL0123.tst".

*Місцезнаходження тесту* – повна назва шляху, розташування на диску ЕОМ.

*Технічна вимога, що тестується* - унікальний ідентифікатор, який відображається в документах технічних вимог.

*Мета тестування* - формулювання того, що має досягти тест.

*Конфігурація засобів тестування* - специфікація вводу / виводу, умови випробувань.

*Налаштування на прогін тесту -* процедура подібна методиці тестування. Вона передбачає опис дій тестувальника і очікуваних результатів. Якщо настроювання автоматизовані, це виглядає так: run setupSC03.pl.

*Методика тестування* - опис дій тестувальника і очікуваних результатів.

*Взаємозалежність тестових випадків* – ідентифікація будь-якого тестового випадку. Для того, щоб виконання даного тесту починалося при означених умовах, необхідно здійснити прогін попередніх тестів.

*Очистка тесту* – якщо система була переведена в нестійкий стан або дані були зруйнованими, очистка дозволяє усунути подібні ситуації.

*Для самостійного вивчення*: Поглибити матеріал лекції за наданою літературою. Вивчення лекційного матеріалу та додаткових джерел. Розгляд запитань і виконання завдань для самостійної роботи, запропонованих на лекції.

*Контрольні запитання для самоперевірки*.

1. Що таке тестування?
2. Що таке відмовостійкість?
3. Які види тестування ви знаєте?
4. В чому полягає суть тестування?
5. В яких моделях немає тестування?
6. Чому при тестуванні необхідно бажати, щоб програма дала збій?
7. Поясніть, чому тестування є деструктивним процесом
8. Що найчастіше тестується в програмних засобах?
9. За якими ознаками прийнято робити класифікацію видів тестування?
10. Які види тестування, з наведених вище, потрібні для ПЗ Вашого проекту?
11. В чому полягає різниця між функціональним і не функціональним тестуванням?
12. Дайте характеристику тестуванню навантаження?
13. Що входить до тестування, що пов'язане зі змінами?
14. В чому відмінність димового та санітарного тестування?
15. В чому особливості тестування установки?
16. В чому особливості регресійного тестування?
17. Назвіть рівні тестування і їх основні характеристики.
18. В чому різниця між компонентним і системним тестуванням?
19. Які підходи до системного тестування, ви знаєте?
20. Які підходи до інтеграційного тестування, ви знаєте?
21. Назвіть вимоги створення тестових випадків.

*Література*

1. Тамре Л. Введение в тестирование программного обеспечения / Л. Тамре. –М.: Издательский дом «Вильямс», 2003. –368 с.
2. Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие / Д.Сайкс, Д.Макгрегор. –К.: Диасофт, 2002. –432 с.
3. Фолк Д. Тестирование программного обеспечения / Д.Фолк, Е.К. Нгуен, С.Канер. – К.: Диасофт, 2003. –400 с.
4. Блэк Р. Ключевые процессы тестирования / Р.Блэк. – М.: Лори, 2006. –544 с.
5. Винниченко И.В. Автоматизация процессов тестирования / И.В.Винниченко. – СПб.: Питер, 2005. –208 с.
6. Криспин Л. Гибкое тестирование. Практическое руководство для тестировщиков ПО и гибких команд / Л.Криспин, Д.Грегори. –М.: Издательский дом «Вильямс», 2010. –464 с.
7. Майерс Г. Искусство тестирования программ / Пер. с англ. под ред. Б. А.Позина. – М.: Финансы и статистика, 1982. – 176 с.
8. Ален И., Голуб. С и С++. Правила программирования. – М.: БИНОМ, 1996. – 272ºс.
9. Дал У., Дейкстра Э., Хоор К. Структурное программирование. – М.: Мир, 1973. – 247º с.
10. 6. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978. – 275 с.
11. Йодан Э.. Структурное проектирование и конструирование програм. – М.: Мир, 1979. – 415 с.