**Лекція 02. Процес тестування програмного забезпечення**

**Тестування –** це метод виявлення помилок у ПС шляхом виконання вихідного коду на тестових даних, збирання робочих характеристик у динаміці виконання в конкретному операційному середовищі, виявлення різних помилок, дефектів, відмовлень і збоїв, викликаних нерегулярними, аномальними ситуаціями або аварійним припиненням роботи системи. Його можна розглядати, як процес семантичного налагодження (перевірки) програми, що полягає у виконанні послідовності різних наборів контрольних тестів, для яких заздалегідь відомий результат. Тобто тестування припускає виконання програми й одержання конкретних результатів виконання тестів.

* Дуже важливо чітко розрізняти причину несправності, для яких буде використовуватися термін несправності або **дефекту,** і небажаний ефект буде спостерігатися в наданих системних послугах, що буде названо **відмовою**.
* Тестування дозволяє виявити відмови, але це дефекти, які можуть і повинні бути видалені.
* Слід визнати, що причина відмови не завжди може бути одразу ідентифікована.

Тести підбираються так, щоб вони охоплювали як найбільше типів ситуацій алгоритму програми. Менш тверда вимога – виконання хоча б один раз кожної гілки програми. Критерій вибору тесту повинен надавати можливість рішення щодо зупинки тестування.

Тестування це спостереження за виконанням зразка програми. В тестуванні для визначення дефекту вдалий тест той, котрий спричиняє збій у системі. Інший вид тестування - це демонстрація того, що програми відповідають своїм специфікаціям або іншим бажаним властивостям, в цьому випадку тестування вдале якщо спостерігаються незначні збої або їх повна відсутність.

Узагальнимо мету тестування:

Основна полягає у визначенні:

* дефектів
* функціональної придатності

Додаткова полягає у визначенні:

* зручності застосування,
* продуктивності та інших характеристик.

Основні підходи до виконання тестування

* деструктивний (негативне, руйнівне тестування)
* конструктивний (позитивне або демонстраційне).

Історично першим видом тестування було налагодження.

*Налагодження* – це перевірка опису програмного об'єкта на мові програмування з метою виявлення в ньому помилок і подальшого їхнього усунення. Помилки виявляються компіляторами при їхньому синтаксичному контролі. Після цього проводиться верифікація з перевірки правильності функцій коду і валідація з перевірки відповідності продукту заданим вимогам.

Мета тестування – перевірка виконання реалізованих функцій відповідно до їхньої специфікації. На основі зовнішніх специфікацій функцій і проектної інформації на процесах ЖЦ створюються функціональні тести, за допомогою яких проводиться тестування з урахуванням вимог, сформульованих на процесі аналізу предметної області. Методи функціонального тестування підрозділяються на статичні і динамічні.

**Статичні методи тестування**

*Статичні методи* використовуються при проведенні інспекцій і розгляді специфікацій компонентів без їхнього виконання.

Техніка статичного аналізу полягає в методичному перегляді (або огляді) і аналізі структури програм, а також у доведенні їхньої правильності вручну за столом. Статичний аналіз направлений на аналіз документів, розроблених на всіх процесах ЖЦ і полягає в інспекції вхідного коду і наскрізного контролю програми.

*Інспекція ПС* – це статична перевірка відповідності програми заданим специфікаціями, проводиться шляхом аналізу різних представлень результатів проектування (документації, вимог, специфікацій, схем або коду програм) на процесах ЖЦ. Перегляди й інспекції результатів проектування і відповідності їх вимогам замовника забезпечують більш високу якість створюваних ПС.

При інспекції програм розглядаються документи робочого проектування на процесах ЖЦ разом з незалежними експертами й учасниками розробки ПС. На початковому процесі проектування інспекція припускає перевірку повноти, цілісності, однозначності, несуперечності і сумісності документів з вимогами до програмної системи. На процесі реалізації системи під *інспекцією* розуміють аналіз текстів програм на дотримання вимог стандартів і прийнятих керівних документів технології програмування. Ефективність такої перевірки полягає в тому, що залучені експерти намагаються подивитися на проблему «з боку» і піддають її всебічному критичному аналізу.

Ці прийоми дозволяють на більш ранніх процесах проектування знайти помилки або недоробки шляхом багаторазового перегляду вхідного опису програми.

**Динамічні методи тестування**

*Динамічні методи тестування* використовуються в процесі виконання програм. Вони базуються на графовій структурі, що пов'язує причини помилок з очікуваними реакціями на них. У процесі тестування накопичується інформація про помилки, що використовується при оцінці показників надійності і якості ПС.

Динамічне тестування орієнтоване на перевірку коректності ПС на множині тестів, що проганяються по ПС, з урахуванням зібраних даних на процесах ЖЦ, проведення виміру окремих показників (число відмов, збоїв) тестування для оцінки характеристик якості, зазначених у вимогах, шляхом виконання системи на ЕОМ.

Тестування ґрунтується на систематичних, статистичних, (імовірнісних) і

імітаційних методах.

Систематичні методи тестування поділяються на методи, у яких програма розглядається як «чорна скринька» (використовується інформація про розв'язувану задачу), і методи, у яких програма розглядається як «біла скринька» з використанням структури програми. Цей вид називають тестуванням з керуванням за даними або керуванням на вході-виході. Ціль – з'ясування обставин, при яких поводження програми не відповідає її специфікації. При цьому кількість виявлених помилок у програмі є критерієм якості тестування.

Ціль динамічного тестування програм за принципом «чорної скриньки» – виявлення одним тестом максимального числа помилок з використанням невеликої підмножини можливих вхідних даних.

*Методи «чорної* скриньки» забезпечують:

– еквівалентне розбиття;

– аналіз граничних значень;

– застосування функціональних діаграм, що в поєднанні з реверсивним аналізом дають досить повну інформацію про функціонування тестованої програми.

Еквівалентна розбивка складається з розділу вхідної області даних програми на скінченне число класів еквівалентності так, щоб кожен тест, що є представником деякого класу, був еквівалентний будь-якому іншому тесту цього класу. Класи еквівалентності виділяються шляхом перебору вхідних умов і розбивки їх на дві групи або більше. При цьому розрізняють два типи класів еквівалентності: правильні, що задають вхідні дані для програми, і неправильні, засновані на завданні помилкових вхідних значень. Розроблення тестів методом еквівалентного розбиття здійснюється в два етапи: виділення класів еквівалентності і побудова тестів.

Методи тестування за принципом «чорної скриньки» використовуються для тестування реалізованих у програмі функцій шляхом перевірки невідповідності між реальною поведінкою функцій і очікуваною поведінкою з урахуванням специфікацій вимог. Під час підготовки до цього тестування будуються таблиці умов, причинно-наслідкового графа й області розбиття. Крім того, готуються тестові набори, що враховують параметри й умови середовища, які впливають на поведінку функцій. Для кожної умови визначається множина значень і обмежень предикатів, за якими тестується програма.

*Метод «білої скриньки»* дозволяє досліджувати внутрішню структуру програми, при чому виявлення всіх помилок у програмі є критерієм вичерпного тестування маршрутів потоків (графа) передач керування, серед яких розглядають:

а) критерій покриття операторів – набір тестів у сукупності повинен забезпечити проходження кожного оператора не менше ніж один раз;

б) критерій тестування областей (відомий як покриття рішень або переходів) – набір тестів у сукупності повинен забезпечити проходження кожної гілки і виходу, принаймні, один раз.

Критерій «б» відповідає простому структурному тесту і найбільш розповсюджений на практиці. Для задоволення цього критерію необхідно побудувати систему шляхів, що містить у собі усі області програми. Перебування такого оптимального покриття в деяких випадках здійснюється просто, а в інших є більш складною задачею.

Тестування за принципом «білої скриньки» орієнтовано на перевірку проходження всіх гілок програм за допомогою застосування шляхового й імітаційного тестування.

*Шляхове тестування* застосовується на рівні модулів і графової моделі програми з вибором тестових ситуацій, підготовки даних і містить у собі тестування наступних елементів:

– операторів, що повинні бути виконані хоча б один раз, без обліку помилок, що можуть залишитися в програмі через велику кількість логічних шляхів і необхідності проходження підмножин цих шляхів;

– шляхів по заданому графу потоків керування для виявлення різних маршрутів передачі керування за допомогою шляхових предикатів, для обчислення якого створюється набір тестових даних, що гарантують проходження всіх шляхів. Проте усі шляхи протестувати неможливо, тому залишаються не виявлені помилки, що можуть виявитися в процесі експлуатації;

– блоків, що розділяють програми на окремі дрібні блоки, які виконуються хоча б один раз або багаторазово при проходженні через шляхи програми, що вміщують сукупність операторів реалізації однієї функції, або на вхідній множині даних, що буде використовуватися при виконанні зазначеного шляху.

«Біла скринька» базується на структурі програми, у випадку ж «чорної скриньки» про структуру програми нічого невідомо. Для виконання тестування за допомогою цих «скриньок» відомими вважаються виконувані функції, входи (вхідні дані) і виходи (вихідні дані), а також логіка обробки програми і опису документації.

**Функціональне тестування**

Мета функціонального тестування – виявлення невідповідностей між реальною поведінкою реалізованих функцій і очікуваною поведінкою відповідно до специфікації і вимог. Функціональні тести повинні охоплювати всі реалізовані функції з урахуванням найбільш ймовірних типів помилок. Тестові сценарії, що поєднують окремі тести, орієнтовані на перевірку якості розв'язку функціональних задач.

Функціональні тести створюються за зовнішніми специфікаціями функцій, проектної інформації і за текстом на мові програмування, що стосуються його функціональних характеристик і застосовуються на процесі комплексного тестування й іспитів для визначення повноти реалізації функціональних задач і їхньої відповідності вхідним вимогам.

До задач функціонального тестування належать:

– ідентифікація множини функціональних вимог;

– ідентифікація зовнішніх функцій і побудова послідовностей функцій відповідно до їхнього використання в ПС;

– ідентифікація множини вхідних даних кожної функції і визначення областей їхньої зміни;

– побудова тестових наборів і сценаріїв тестування функцій;

– виявлення і подання усіх функціональних вимог за допомогою тестових наборів і проведення тестування помилок у програмі і при взаємодії із середовищем.

Тести, створювані за проектною інформацією, пов'язані зі структурами даних, алгоритмами, інтерфейсами між окремими компонентами і застосовуються для тестування компонентів і їхніх інтерфейсів. Основна мета – забезпечення повноти і погодженості реалізованих функцій і інтерфейсів між ними.

В основу комбінованого методу «чорної скриньки» і «білої скриньки» покладено розбивку вхідної області функції на підобласті виявлення помилок. Підобласть містить у собі однорідні елементи, які обробляються коректно або некоректно. Для тестування підобласті застосовується виконання програми на одному з елементів цієї області.

Передумови функціонального тестування:

– коректне оформлення вимог і обмежень до якості ПС;

– коректний опис моделі функціонування ПС у середовищі експлуатації замовника;

– адекватність моделі ПС заданому класу.

**Інфраструктура перевірки правильності програмних систем**

Під *інфраструктурою* перевірки правильності (доведення, верифікації і тестування) програмних систем розуміють інтегрований набір загальнодоступних технічних, технологічних і методологічних ресурсів, що знаходяться у розпорядженні команди розробників, верифікаторів і тестувальників, які виконують роботи з розроблення правильної системи за договорами із організаціями-замовниками.

Команда виконує дії з підготовки і проведення таких задач:

– виділення об'єктів перевірки на процесах ЖЦ та на завершальному процесі тестування;

– аналіз і класифікація помилок для розглянутого класу програм, що перевіряються;

– підготовка даних для верифікації правильності виконання функцій;

– підготовка даних і тестів для їхнього виконання і пошуку різного роду помилок і відмовлень у компонентах і в системі в цілому;

– розроблення завдань підгрупами команди з проведення і керування процесом досягнення правильної програми;

– підготовка до перевірки виконання вимог до ПС і до системи;

– аналіз результатів верифікації і тестування системи, отриманих підгрупами.

Об'єкти процесу – компоненти, групи компонентів, підсистеми і система. Для кожного з них формується стратегія проведення верифікації і тестування. Якщо об'єкт готовий і належить до «білої скриньки» або до «чорної скриньки», склад компонентів якого невідомий, то верифікація функцій проводиться зі спеціально підготовленими даними перевірки функцій і вхідних тестових даних для тестування й отримання вихідних даних. Головна мета верифікації перевірити правильність виконання функцій.

Стратегічна мета тестування – переконатися, що кожен розглянутий вхідний набір даних відповідає очікуваним вихідних даним. Проектувальник тестів повинен заглянути усередину «чорної скриньки» і дослідити деталі процесів обробки даних, питання забезпечення захисту і відновлення даних, а також інтерфейси з іншими програмами і системами. Це сприяє підготовці тестових даних для проведення тестування. Для деяких типів об'єктів група інженерії тестування не може згенерувати представницьку безліч тестових наборів, що демонстрували б функціональну правильність роботи компонентів при всіх можливих наборах тестів.

Тому кращим є метод «білої скриньки», при якому можна використовувати логічну структуру об'єкта для організації перевірки програми в різних її областях. Наприклад, можна виконати верифікацію функцій і підготувати тестові набори, що проходять через всі оператори або всі контрольні точки компонента для того, щоб переконатися в отриманні правильних результатів.

**Класифікація помилок і методи їхнього пошуку**

Міжнародний стандарт ANSI/IEEE–729–83 розділяє всі помилки в розробці програм на такі типи.

***Помилка* (error)** *–* стан програми, при якому видаються неправильні результати, причиною яких є недоліки (flaw) в операторах програми або в технологічному процесі її розроблення, що приводить до неправильної інтерпретації вихідної інформації, отже, і до невірного розв’язку.

***Дефект* (fault)** у програмі – наслідок помилок розробника на кожному з процесів проектування, що може утримуватися у вхідних або проектних специфікаціях, текстах кодів програм, експлуатаційній документації тощо. У процесі виконання програми можуть бути виявлені дефект або збій.

***Відмова* (failure)** – це відхилення програми від функціонування або неможливість програми виконувати функції, визначені вимогами й обмеженнями, що розглядається як подія, яка сприяє переходу програми в непрацездатний стан через помилки, приховані у ній дефекти або збої у середовищі функціонування. Відмова може бути за таких причин:

– помилкова специфікація або пропущена вимога, яка означає, що специфікація точно не відбиває того, що припускав користувач;

– специфікація може містити у собі вимогу, яку неможливо виконати на даній апаратурі і програмному забезпеченні;

– проект програми може містити у собі помилки (наприклад, база даних спроектована без засобів захисту від несанкціонованого доступу користувача, а потрібен захист);

– програма може бути неправильною, тобто вона виконує невластивий алгоритм або він реалізований не цілком.

Таким чином, відмова, як правило, є результатами однієї помилки або більше у програмі, а також наявності різного роду дефектів.

**Загальні класи помилок.** Усі помилки, що виникають у програмах, розподіляють на такі класи:

– логічні і функціональні помилки;

– помилки обчислень і часу виконання;

– помилки вводу-виводу і маніпулювання даними;

– помилки інтерфейсів;

– помилки обсягу даних і ін.

*Логічні помилки* – наслідок порушення логіки алгоритму, внутрішньої непогодженості змінних і операторів, а також мовних правил програмування.

*Функціональні помилки* – наслідок неправильно визначених функцій, порушення порядку їхнього застосування або відсутності повноти їхньої реалізації тощо.

*Помилки обчислень* виникають через неточність вхідних даних і реалізованих формул, похибок методів, неправильного застосування операцій обчислень.

*Помилки часу виконання* зв'язані з відсутністю необхідної швидкості обробки запитів, або часу виконання або відновлення програми.

*Помилки вводу-виводу* і маніпулювання даними є наслідком неякісної підготовки даних для виконання програми, збоїв при занесенні їх у базу даних або при вибірці з неї.

*Помилки інтерфейсу* належать до помилок взаємозв'язку окремих елементів одного з одним, що виявляється при передачі даних між ними, а також при взаємодії із середовищем функціонування.

*Помилки обсягу* належать до даних і є наслідком того, що реалізовані методи доступу і розміри баз даних не задовольняють реальні обсяги інформації системи або інтенсивності їхньої обробки.

Наведені основні класи помилок властиві різним типам компонентів ПС і виявляються вони в програмах по-різному. Так, при роботі з БД виникають помилки подання і маніпулювання даними, логічні помилки в завданні прикладних процедур обробки даних та ін. У програмах обчислювального характеру переважають помилки обчислень, а в програмах керування й обробки – логічні і функціональні помилки. У ПС, що складається з багатьох різномовних програм, які реалізують різні функції, можуть міститися помилки різних типів. Помилки інтерфейсів і порушення обсягу характерні для будь-якого типу ПС.

Аналіз типів помилок у програмах є необхідною умовою створення планів і методів тестування для забезпечення правильності ПС. На сучасному процесі розвитку засобів підтримки розробки ПС (CASE–технології, інструменти) при проектуванні ПС захищається від найбільш типових помилок і запобігається поява дефектів.

*Фірма IВМ* розробила підхід до класифікації помилок, названий ортогональною класифікацією дефектів. При такому підході розподіл помилок за категоріями робить відповідальний розробник. Класифікація не залежить від продукту, організації розробки, вона може застосуються до всіх процесів розроблення ПС різного призначення. Відповідно до даної класифікації в табл. 1 наведено список помилок.

Таблиця 1. Ортогональна класифікація дефектів IBM

|  |  |
| --- | --- |
| **Контекст**  **помилки** | **Класифікація дефектів** |
| Функція | Помилки інтерфейсів кінцевих користувачів ПС, викликані апаратурою або зв'язані з зовнішніми структурами даних |
| Інтерфейс | Помилки у взаємодії з іншими компонентами, у викликах, макросах, що керують блоками або в списку параметрів |
| Логіка | Помилки в програмній логіці, неохопленій валідацією, а також у використанні значень змінних |
| Присвоювання | Помилки в структурі даних або в ініціалізації змінних окремих частин програми |
| Зациклення | Помилки, викликані ресурсом часу, реальним часом або розподілом часу |
| Середовище | Помилки в репозитарії, у керуванні змінами або в контрольованих версіях проекту |
| Алгоритм | Помилки, пов'язані з забезпеченням ефективності, коректності алгоритмів або структур дані системи |
| Документація | Помилки в записах документів супроводу або в публікаціях |

Передбачено ситуації, коли знайдена неініційована змінна або ініційованій

змінній привласнене неправильне значення.

Ортогональність схеми класифікації полягає в тому, що будь-який її термін належить тільки до однієї категорії. Іншими словами, помилка, що простежується в системі, повинна знаходитися в одному з класів, що дає можливість різним розробникам класифікувати помилки однаковим способом.

*Фірма Hewlett–Packard* використовувала класифікацію Буча, встановивши відсоткове співвідношення помилок, що виявляються в ПС на різних стадіях розробки (рис. 1.). Це співвідношення, типове для багатьох фірм, що роблять ПС, має деякі відхилення від інших.

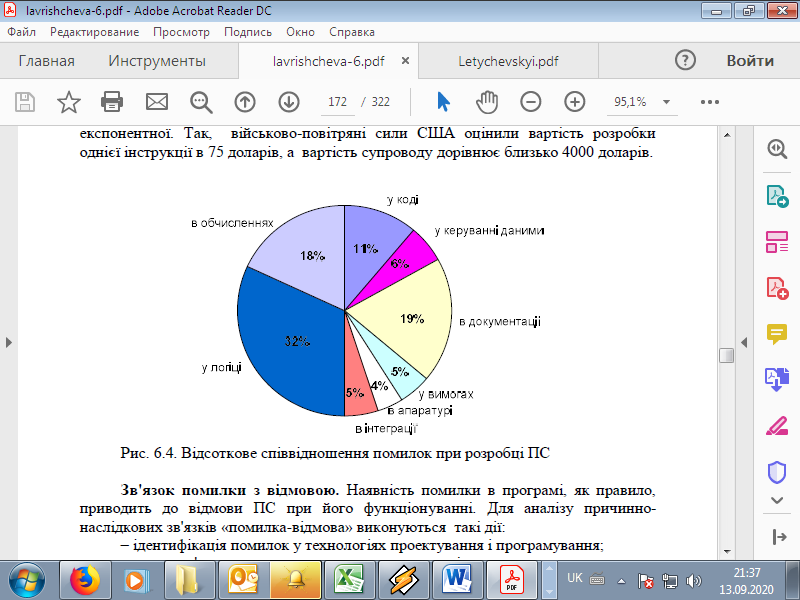


Рис. 1. Відсоткове співвідношення помилок при розробці ПС

В середньому вартість аналізу і формування вимог, внесення до них змін становить приблизно 10%, аналогічно оцінюється вартість специфікації продукту. Вартість кодування оцінюється більше ніж 20%, а вартість тестування продукту дорівнює більш ніж 45% його загальної вартості. Значну частину вартості становить супровід готового продукту і виправлення виявлених у ньому помилок.

Дослідження фірм IBM показали, чим пізніше виявляється помилка в програмі, тим дорожче коштує її виправлення, ця залежність близька до експонентної.

**Зв'язок помилки з відмовою.** Наявність помилки в програмі, як правило, приводить до відмови ПС при його функціонуванні. Для аналізу причинно- наслідкових зв'язків «помилка-відмова» виконуються такі дії:

– ідентифікація помилок у технологіях проектування і програмування;

– взаємозв'язок помилок процесу проектування і помилок, що допускаються людиною;

– класифікація відмов, помилок і можливих помилок, а також дефектів на кожному процесі розробки;

– зіставлення помилок людини, що допускаються на визначеному процесі розробки, і дефектів в об'єкті, як наслідків помилок специфікації проекту, моделей програм;

– перевірка і захист від помилок на всіх процесах ЖЦ, а також виявлення дефектів на кожному процесі розробки;

– зіставлення дефектів і відмовлень у ПС для розробки системи взаємозв'язків і методики локалізації, збирання й аналізу інформації про відмови і дефекти;

– розробка підходів до процесів документування й супроводження ПС.

Кінцева мета причинно-наслідкових зв'язків «помилка-відмова» полягає у визначенні методів і засобів тестування і виявлення помилок визначених класів, а також критеріїв завершення тестування на множині наборів даних; у визначенні шляхів удосконалювання організації процесу розроблення, тестування і супроводу ПС.

Наведемо таку класифікацію типів відмов:

– апаратна, при якому загальносистемне ПС не працездатне;

– інформаційна, викликана помилками у вхідних даних і передачі даних по каналах зв'язку, а також при збоях пристроїв вводу, як наслідок апаратних відмов;

– програмна, при наявності помилок у компонентах системи;

– ергономічна, викликана помилками оператора при його взаємодії з комп’ютером (це відмовлення – вторинна відмова, може привести до інформаційної або функціональної відмови).

Деякі помилки можуть бути, з одного боку, наслідком недоробок при визначенні вимог, проекту, генерації вихідного коду або документації, з іншого боку, вони виникають в процесі розробки програми або інтерфейсів окремих елементів програми (порушення порядку параметрів і т.п.).

Підготовка тестів складається з кроків.

*Крок 1. Дослідження.*

*Крок 2. Проектування тестів.*

*Крок 3. Виконання тестів.*

*Крок 4. Визначення критерію покриття..*

***Крок 1. Дослідження***

* Формування списку функцій (ієрархії функцій).
* Розбивка функцій на основні й другорядні.
* Виявлення областей можливої нестійкості продукту (підданих відмовам функцій і даних).

|  |  |
| --- | --- |
| **Визначення** | **Критерії віднесення** |
| *Основна функція* - будь-яка зовнішня функція, відмови в якій означають невідповідність продукту своєму призначенню. | 1. Функції, що є визначальними для використання продукту по призначенню. 2. Функції, часто використовувані звичайними користувачами в сеансі роботи. 3. Функції, які можуть використовуватися рідко, але їхні відмови приводять до значних негативних наслідків. |
| *Другорядна функція -* функція, без якої продукт може використовуватися. | Рідко використовувані функції, відмови при виконанні яких не можуть мати істотних наслідків. |

Приклади областей можливої нестійкості функцій

* функції обробки зовнішніх подій;
* функції, що інтенсивно використовують оперативна пам'ять;
* дуже складні функції;
* функції, що використовують засоби Windows і/або її параметри, що змінюють (настроювання параметрів);
* функції, що маніпулюють конфігурацією Windows;
* функції аналізу вхідних даних і обробки помилок;
* функції, що підмінюють базові функції Windows (наприклад, відновлення вилучених файлів);
* функції або групи функцій, що використовують багато паралельних процесів;
* функції, що працюють із багатьма файлами одночасно;
* функції, що працюють із файлами, розташованими в мережі. Приклади областей можливої нестійкості при обробці даних:
* *документи:* довгі, багато одночасно відкритих;
* *запису:* довгі, багато записів, порожні, складні;
* *списки:* довгі, порожні або багатоколонкові;
* *поля:* уведення великої кількості символів; дуже більші значення числових полів;
* *об'єкти:* багато, великі, складні й ін.

***Крок 2. Проектування тестів.***

| **Визначення мети** | **Критерій проходу** | **Критерій відмови** |
| --- | --- | --- |
| *Функціональність* - здатність продукту виконувати необхідні функції. | Функція виконується відповідно до її призначення. | Хоча б одна функція не виконується відповідно до її призначення. |
| Будь-яке неправильне виконання функції не приводить до серйозних наслідків при коректному використанні. | Неправильне виконання приводить до серйозних наслідків навіть при коректному використанні. |
| *Стійкість* - здатність продукту функціонувати без відмов при підвищених навантаженнях (за часом, пам'яті). | Робота продукту при підвищених навантаженнях не приводить до руйнування Windows.  Робота продукту не приводить до втрати даних, зависанню, відмові. При тестуванні не було виявлено відмов або будь-яких дефектів при виконанні основних функцій. | Робота продукту руйнує Windows.  Робота продукту може привести до втрати даних, зависанню, відмові. При тестуванні хоч однієї з основних функцій спостерігалися відмови. |

***Крок 3. Виконання тестів***

Завдання:

* тестування всіх основних функцій;
* тестування ідентифікованих областей потенційної нестійкості;
* вибіркове тестування другорядних функцій;
* реєстрація відмов;
* фіксація будь-яких виявлених проблем (для подальшого вивчення).

***Крок 4. Визначення критерію покриття***

* протестовані всі основні функцій;
* протестовані обрані другорядні функції;
* протестовані обрані області можливої нестійкості (п'ять - десять функцій, , які можуть привести до нестійкої роботи продукту, протестувати їх на тестових даних).

Розподіл часу

* 80% часу - на основні функції,
* 10% - на другорядні й
* 10% - на області потенційної нестійкості.

**Еквівалентна розбивка**

Критерії :

* тести включають значення тих самих вхідних даних;
* при запуску тестів виконуються ті самі операції;
* від виконання тестів очікуються однакові результати;
* або жоден з тестів не викликає виконання блоку обробки помилок, або всі тести викликають виконання цього блоку (у припущенні, що програма взагалі містить блоки обробки помилок).

Приклад опису класів еквівалентності

|  |  |  |
| --- | --- | --- |
| **Вхідні події** | **Припустимі класи еквівалентності** | **Неприпустимі класи еквівалентності** |
| Уведення числа | Числа від 0 до 9000 | Числа менше 0 і числа більше 9000  Пробіл  Нечислові символи  Вираз, що обчислюється, результат обчислення якого перебуває поза припустимим інтервалом |
| Уведення першої букви прізвища | Перший символ повинен бути заголовною буквою | Перший символ мала літера  Перший символ - не буква |

**Розбивка вхідного простору на категорії** виконується у послідовності

1. *Декомпозиція функції на функціональні елементи, які можуть тестуватися незалежно.*
2. *Ідентифікація параметрів і умов середовища, що впливають на поводження функції.*
3. *Виділення категорій інформації, що характеризує кожний параметр і умова середовища.*
4. *Розбивка кожної категорії на чіткі альтернативи, які включають різні види значень, можливі для категорії.*
5. *Формування формальної специфікації тесту для кожного функціонального елемента.*

Приклад формування формальної специфікації тесту для кожного функціонального елемента

|  |  |  |
| --- | --- | --- |
| **Параметр** | **Категорія** | **Альтернативи** |
| Поле вибору | Зміст | Текст  Піктограма  Значок з текстом |
|  | Стан | Поле не зазначене/обране  Поле не зазначене/не обране  Поле зазначене/обране  Поле зазначене/не обране |
|  | Тип поля | Однозначний вибір  Багатозначний вибір  Розширений вибір |
| Меню | Спосіб вибору | Вибір курсором  Виділена буква/(комбінація букв)  Функціональна клавіша  Вибір за допомогою клавіатури |

Що надає такий підхід:

* дозволяє охопити відразу обоє основних аспекту тестування - перевірку повноти реалізації функцій і виявлення різних класів дефектів у найбільш уразливих частинах програми;
* функціональний аналіз є невід'ємною частиною методу й виконується паралельно зі специфікацією функціональних вимог (або відразу після), забезпечуючи своєчасне усунення дефектів специфікацій;
* хоча специфікація тесту повинна охоплювати всі можливі категорії інформації й варіанти сполучення альтернатив, - застосовуючи механізм обмежень і керуючись практичними міркуваннями, можна управляти обсягом тестування;
* процес перебору альтернатив і «відсівання» неможливих або небажаних їхніх сполучень може бути автоматизований, що позбавить тестувальника від рутинної роботи.

***Контрольні запитання для самоперевірки***.

1. Надайте визначення тестування. В чому полягає його мета?
2. За якими принципами підбираються тести? Наведіть приклад.
3. В чому полягає використання методів статичного тестування?
4. В чому полягає динамічне тестування? Визначте основні методи.
5. Чи обов’язково використовувати обидва методи "білої" та "чорної скрині" для тестування ПЗ? Хто на Вашу думку повинен виконувати таке тестування?
6. В чому мета функціонального тестування?
7. Що мається на увазі під *інфраструктурою* перевірки правильності програмних систем?
8. Як міжнародний стандарт ANSI/IEEE–729–83 поділяє помилки в ПЗ?
9. Визначте класи помилок.

**Рекомендована література**

1. Бабенко Л.П., Лавріщева К.М*.* Основи програмної інженерії.– Навч. посібник.–К.: Знання, 2001.– 269с.
2. И. Соммервиль. Инженерия программного обеспечения,  
    6 изд. – И.д. "Вильямс", 2002.
3. Лавріщева К.М. Програмна інженерія. Електронний підручник: http://csc.knu.ua/uk/library/books/lavrishcheva-6.pdf
4. Лавріщева К.М. Програмна інженерія. – Підручник.–К.:Академперіодика, 2008.–415с.
5. Тамре Л. Введение в тестирование программного обеспечения / Л. Тамре. –М.: Издательский дом «Вильямс», 2003. –368 с.
6. Сайкс Д. Тестирование объектно-ориентированного программного обеспечения. Практическое пособие / Д.Сайкс, Д.Макгрегор. –К.: Диасофт, 2002. –432 с.
7. Фолк Д. Тестирование программного обеспечения / Д.Фолк, Е.К. Нгуен, С.Канер. – К.: Диасофт, 2003. –400 с.

***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.