**Лабораторна робота № 7.**

***Тема****:* **“Тестування і відладка програмного засобу”.**

***Мета:*** Розглянути суть та методи створення тестів для перевірки працездатності програми.

***Теоретичні відомості:***

**Основні поняття.**

*Відладка* ПС − це діяльність, направлена на виявлення і виправлення помилок в ПС з використанням процесів виконання його програм. *Тестування* ПС − це процес виконання його програм на деякому наборі даних, для якого наперед відомий результат застосування або відомі правила поведінки цих програм. Вказаний набір даних називається *тестовим або просто тестом*. Таким чином, відладку можна представити у вигляді багатократного повторення трьох процесів: тестування, в результаті якого може бути констатовано наявність в ПС помилки, пошуку місця помилки в програмах і документації ПС і редагування програм і документації з метою усунення виявленої помилки. Іншими словами:

Відладка = Тестування + Пошук помилок + Редагування.

У зарубіжній літературі відладку часто розуміють тільки як процес пошуку і виправлення помилок (без тестування), факт наявності яких встановлюється при тестуванні. Іноді тестування і відладку вважають синонімами. У нашій країні в поняття відладки звичайно включають і тестування, тому ми слідуватимемо традиції, що склалася. Втім, сумісний розгляд в даній лекції цих процесів робить вказане різночитання не таким істотним. Слід, проте, відзначити, що тестування використовується і як частина процесу атестації ПС.

**Принципи і види відладки програмного засобу.**

Успіх відладки ПС в значній мірі зумовлює раціональна організація тестування. При відладці ПС відшукуються і усуваються, в основному, ті помилки, наявність яких в ПС встановлюється при тестуванні. Як було вже відмічено, тестування не може довести правильність ПС, в кращому разі воно може продемонструвати наявність в ньому помилки. Іншими словами, не можна гарантувати, що тестуванням ПС практично здійснимим набором тестів можна встановити наявність тієї, що кожної є в ПС помилки. Тому виникає два завдання. Перше завдання: підготувати такий набір тестів і застосувати до них ПС, щоб виявити в ньому по можливості більше число помилок. Проте чим довше продовжується процес тестування (і відладки в цілому), тим більшою стає вартість ПС. Звідси друге завдання: визначити момент закінчення відладки ПС (або окремої його компоненти). Ознакою можливості закінчення відладки є повнота обхвату пропущеними через ПС тестами (тобто тестами, до яких застосовано ПС) безлічі різних ситуацій, що виникають при виконанні програм ПС, і відносно рідкісний прояв помилок в ПС на останньому відрізку процесу тестування. Останнє визначається відповідно до необхідного ступеня надійності ПС, 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 часу, відведеному на тестування) виявляти більше число помилок в ПС, необхідно, по-перше, наперед планувати цей набір і, по-друге, використовувати раціональну стратегію планування тестів. Проектування тестів можна починати відразу ж після завершення етапу зовнішнього опису ПС. Можливі різні підходи до вироблення стратегії проектування тестів, які можна умовно графічно розмістити (див. мал. 10.1) між наступними двома крайніми підходами. Лівий крайній підхід полягає в тому, що тести проектуються тільки на підставі вивчення специфікацій ПС (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 інакше деякі ділянки програм ПС можуть не працювати при пропуску будь-якого тесту, а це означає, що помилки, що містяться в них, не виявлятимуться. Проте тестування ПС повною безліччю наборів вхідних даних практично нездійсненно. Правий крайній підхід полягає в тому, що тести проектуються на підставі вивчення текстів програм з метою протестувати всі шляхи виконання кожної програм ПС. Якщо взяти до уваги наявність в програмах циклів із змінним числом повторень, то різних шляхів виконання програм ПС може опинитися також надзвичайно багато, так що їх тестування також буде практично нездійсненно.



Мал. 1. Спектр підходів 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, але вона вимагає також проектування деяких тестів і по текстах програм. При цьому в першому випадку ця стратегія базується на принципах:

* 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 кожну особливу (виняткову) ситуацію, вказану в специфікаціях, − хоч би один тест.

У другому випадку ця стратегія базується на принципі: кожна команда кожної програми ПС повинна пропрацювати хоч би на одному тесті.

Оптимальну стратегію проектування тестів можна конкретизувати на підставі наступного принципу: для кожного програмного документа (включаючи тексти програм), що входить до складу ПС, повинні проектуватися свої тести з метою виявлення в ньому помилок. В усякому разі, цей принцип необхідно дотримувати відповідно до визначення ПС і змістом поняття технології програмування як технології розробки надійних ПС (див. лекцію 1). У зв'язку з цим Майерс навіть визначає різні види тестування залежно від виду програмного документа, на підставі якого будуються тести. У нашій країні розрізняються два основні види відладки (включаючи тестування): автономну і комплексну відладку ПС. *Автономна* відладка ПС означає послідовне роздільне тестування різних частин програм, що входять в ПС, з пошуком і виправленням в них помилок, що фіксуються при тестуванні. Вона фактично включає відладку кожного програмного модуля і відладку сполучення модулів. Комплексна відладка означає тестування ПС в цілому з пошуком і виправленням помилок, що фіксуються при тестуванні, у всіх документах (включаючи тексти програм ПС), що відносяться до ПС в цілому. До таких документів відносяться визначення вимог до ПС, специфікації якості ПС, функціональна специфікація ПС, опис архітектури ПС і тексти програм ПС.

**Заповіді відладки програмного засобу.**

Але спочатку слід зазначити деякий феномен, який підтверджує важливість попередження помилок на попередніх етапах розробки: у міру зростання числа виявлених і виправлених помилок в ПС *росте* також відносна вірогідність існування в ньому невиявлених помилок. Це пояснюється тим, що при зростанні числа помилок, виявлених в ПС, уточнюється і наше уявлення про загальне число допущених в ньому помилок, а значить, в якійсь мірі, і про число невиявлених ще помилок.

Нижче приводяться рекомендації по організації відладки у формі заповідей.

*Заповідь 1*. Вважайте тестування ключовим завданням розробки ПС, доручайте його найкваліфікованішим і обдарованим програмістам; небажано тестувати свою власну програму.

*Заповідь 2*. Хороший той тест, для якого висока вірогідність виявити помилку, а не той, який демонструє правильну роботу програми.

*Заповідь 3*. Готуйте тести як для правильних, так і для неправильних даних.

*Заповідь 4*. Документуйте пропуск тестів через комп'ютер; детально вивчайте результати кожного тесту; уникайте тестів, пропуск яких не можна повторити.

*Заповідь 5.* Кожен модуль підключайте до програми тільки один раз; ніколи не змінюйте програму, щоб полегшити її тестування.

*Заповідь 6.* Пропускайте наново всі тести, пов'язані з перевіркою роботи якої-небудь програми ПС або її взаємодії з іншими програмами, якщо в неї були внесені зміни (наприклад, в результаті усунення помилки).

**Автономна відладка програмного засобу.**

При автономній відладці ПС кожен модуль насправді тестується в деякому програмному оточенні, крім випадку, коли відладжувана програма складається тільки з одного модуля. Це оточення складається з інших модулів, частина яких є модулями відладжуваної програми, які вже відладжені, а частина − модулями, що управляють відладкою (*налагоджувальними* модулями, див. нижче). Таким чином, при автономній відладці тестується завжди деяка програма (*тестована програма*), побудована спеціально для тестування відладжуваного модуля. Ця програма лише частково співпадає з відладжуваною програмою, крім випадку, коли відладжується останній модуль відладжуваної програми. В процесі автономної відладки ПС проводиться нарощування тестованої програми відладженими модулями: при переході до відладки наступного модуля в його програмне оточення додається останній відладжений модуль. Такий процес нарощування програмного оточення відладженими модулями називається *інтеграцією* програми. Налагоджувальні модулі, що входять в оточення відладжуваного модуля, залежать від порядку, в якому відладжуються модулі цієї програми, від того, який модуль відладжується і, можливо, від того, який тест пропускатиметься.

При висхідному тестуванні це оточення міститиме тільки один налагоджувальний модуль (крім випадку, коли відладжується останній модуль відладжуваної програми), який буде головним в тестованій програмі. Такий налагоджувальний модуль називають *ведучим* (або драйвером). Провідний налагоджувальний модуль готує інформаційне середовище для тестування відладжуваного модуля (тобто формує її стан, потрібний для тестування цього модуля, зокрема, шляхом введення деяких тестових даних), здійснює звернення до відладжуваного модуля і після закінчення його роботи видає необхідні повідомлення. При відладці одного модуля для різних тестів можуть складатися різні ведучі налагоджувальні модулі.

При низхідному тестуванні оточення відладжуваного модуля як налагоджувальні модулі містить *налагоджувальні імітатори* (заглушки) деяких ще не відладжених модулів. До таких модулів відносяться, перш за все, всі модулі, до яких може звертатися відладжуваний модуль, а також ще не відладжені модулі, до яких можуть звертатися вже відладжені модулі (включені в це оточення). Деякі з цих імітаторів при відладці одного модуля можуть змінюватися для різних тестів.

На практиці в оточенні відладжуваного модуля можуть міститися налагоджувальні модулі обох типів, якщо використовується змішана стратегія тестування. Це пов'язано з тим, що як висхідне, так і низхідне тестування має свої достоїнства і свої недоліки.

До *достоїнств висхідного тестування відносяться:*

* простота підготовки тестів
* можливість повної реалізації плану тестування модуля.
* Це пов'язано з тим, що тестовий стан інформаційного середовища готується безпосередньо перед зверненням до відладжуваного модуля (провідним налагоджувальним модулем).
* *Недоліками висхідного тестування* є наступні його особливості:
* тестові дані готуються, як правило, не в тій формі, яка розрахована на користувача (крім випадку, коли відладжується останній, головний, модуль відладжуваної програм);
* великий об'єм налагоджувального програмування (при відладці одного модуля доводиться складати багато провідних налагоджувальних модулів, що формують відповідний стан інформаційного середовища для різних тестів);
* необхідність спеціального тестування сполучення модулів.
* До *достоїнств низхідного тестування* відносяться наступні його особливості:
* більшість тестів готуються у формі, розрахованій на користувача;
* у багатьох випадках відносно невеликий об'єм налагоджувального програмування (імітатори модулів, як правило, вельми прості і кожен придатний для великого числа, нерідко −для всіх, тестів);
* відпадає необхідність тестування сполучення модулів.

*Недоліком низхідного тестування* є те, що тестовий стан інформаційного середовища перед зверненням до відладжуваного модуля готується побічно − воно є результатом застосування вже відладжених модулів до тестових даних або даних, що видаються імітаторами. Це, по-перше, утрудняє підготовку тестів і вимагає високої кваліфікації тестовика (розробника тестів), а по-друге, робить скрутним або навіть неможливим реалізацію повного плану тестування відладжуваного модуля. Вказаний недолік іноді вимушує розробників застосовувати висхідне тестування навіть у разі низхідної розробки. Проте частіше застосовують деякі модифікації низхідного тестування, або деяку комбінацію низхідного і висхідного тестування. Виходячи з того, що низхідне тестування, у принципі, є переважним, зупинимося на прийомах, що дозволяють в якійсь мірі подолати вказані труднощі.

Перш за все, необхідно організувати відладку програми так, щоб якомога раніше були відладжені модулі, що здійснюють введення даних, − тоді тестові дані можна готувати у формі, розрахованій на користувача, що істотно спростить підготовку подальших тестів. Далеко не завжди це введення здійснюється в головному модулі, тому доводиться в першу чергу відладжувати ланцюжки модулів, ведучі до модулів, що здійснюють вказане введення. Поки модулі, що здійснюють введення даних, не відладжені, тестові дані поставляються деякими імітаторами: вони або включаються в імітатор як його частина, або вводяться цим імітатором.

При низхідному тестуванні деякі стани інформаційного середовища, при яких потрібно тестувати відладжуваний модуль, можуть не виникати при виконанні відладжуваної програми ні при яких вхідних даних. У цих випадках можна було б взагалі не тестувати відладжуваний модуль, оскільки помилки, що виявляються при цьому, не виявлятимуться при виконанні відладжуваної програми ні при яких вхідних даних. Проте так поступати не рекомендується, оскільки при змінах відладжуваної програми (наприклад, при супроводі ПС) не використані для тестування відладжуваного модуля стани інформаційного середовища можуть вже виникати, що вимагає додаткового тестування цього модуля (а цього при раціональній організації відладки можна було б не робити, якщо сам даний модуль не змінювався). Для здійснення тестування відладжуваного модуля у вказаних ситуаціях іноді використовують відповідні імітатори, щоб створити необхідний стан інформаційного середовища. Частіше ж користуються модифікованим варіантом низхідного тестування, при якому відладжувані модулі перед їх інтеграцією заздалегідь тестуються окремо (в цьому випадку в оточенні відладжуваного модуля з'являється провідний налагоджувальний модуль, разом з імітаторами модулів, до яких може звертатися відладжуваний модуль). Проте, представляється доцільнішою інша модифікація низхідного тестування: після завершення низхідного тестування відладжуваного модуля для досяжних тестових станів інформаційного середовища слідує його окремо протестувати для решти необхідних станів інформаційного середовища.

Часто застосовують також комбінацію висхідного і низхідного тестування, яку називають методом *сандвіча*. Суть цього методу полягає в одночасному здійсненні як висхідного, так і низхідного тестування, поки ці два процеси тестування не зустрінуться на якому-небудь модулі десь у середині структури відладжуваної програми. Цей метод при розумному порядку тестування дозволяє скористатися достоїнствами як висхідного, так і низхідного тестування, а також в значній мірі нейтралізувати їх недоліки.

Вельми важливим при автономній відладці є тестування

сполучення модулів. Річ у тому, що специфікація кожного модуля програми, окрім головного, використовується в цієї програми в двох ситуаціях: по-перше, при розробці тексту (іноді говорять: тіла) цього модуля і, по-друге, при написанні звернення до цього модуля в інших модулях програми. І в тому, і в іншому випадку в результаті помилки може бути порушено необхідну відповідність заданої специфікації модуля. Такі помилки потрібно виявляти і усувати. Для цього і призначено тестування сполучення модулів. При низхідному тестуванні тестування сполучення здійснюється попутно кожним тестом, що пропускається, що вважають гідністю низхідного тестування. При висхідному тестуванні звернення до відладжуваного модуля проводиться не з модулів відладжуваної програми, а з провідного налагоджувального модуля. У зв'язку з цим існує небезпека, що останній модуль може пристосуватися до деяких "помилок" відладжуваного модуля. Тому, приступаючи (в процесі інтеграції програми) до відладки нового модуля, доводиться тестувати кожне звернення до раніше відладженого модуля з метою виявлення неузгодженості цього поводження з тілом відповідного модуля (і не виключено, що винен в цьому раніше відладжений модуль). Таким чином, доводиться частково повторювати в нових умовах тестування раніше відладженого модуля, при цьому виникають ті ж труднощі, що і при низхідному тестуванні.

Автономне тестування модуля доцільно здійснювати в чотири послідовно виконувані кроки.

Крок 1. На підставі специфікації відладжуваного модуля підготуйте тести для кожної можливості і кожної ситуації, для кожної межі областей допустимих значень всіх вхідних даних, для кожної області зміни даних, для кожної області неприпустимих значень всіх вхідних даних і кожної неприпустимої умови.

Крок 2. Перевірте текст модуля, щоб переконатися, що кожен напрям будь-якого розгалуження буде пройдено хоч би на одному тесті. Додайте бракуючі тести.

Крок 3. Перевірте текст модуля, щоб переконатися, що для кожного циклу існують тести, що забезпечують, принаймні, три наступні ситуації: тіло циклу не виконується жодного разу, тіло циклу виконується один раз і тіло циклу виконується максимальне число раз. Додайте бракуючі тести.

Крок 4. Перевірте текст модуля, щоб переконатися, що існують тести, перевіряючі чутливість до окремих особливих значень вхідних даних. Додайте бракуючі тести.

**Комплексна відладка програмного засобу.**

Як вже було сказано вище, при комплексній відладці тестується ПС в цілому, причому тести готуються по кожному з документів ПС. Тестування цих документів проводиться, як правило, в порядку, зворотному їх розробці. Виняток становить лише тестування документації по застосуванню, яка розробляється по зовнішньому опису паралельно з розробкою текстів програм − це тестування краще проводити після завершення тестування зовнішнього опису. Тестування при комплексній відладці є застосуванням ПС до конкретних даних, які у принципі можуть виникнути у користувача (зокрема, всі тести готуються у формі, розрахованій на користувача), але, можливо, в модельованому (а не в реальній) середовищі. Наприклад, деякі недоступні при комплексній відладці пристрої введення і висновку можуть бути замінені їх програмними імітаторами.

*Тестування архітектури ПС.* Метою тестування є пошук невідповідності між описом архітектури і сукупністю програм ПС. До моменту почала тестування архітектури ПС повинна бути вже закінчена автономна відладка кожної підсистеми. Помилки реалізації архітектури можуть бути зв'язані, перш за все, з взаємодією цих підсистем, зокрема, з реалізацією архітектурних функцій (якщо вони є). Тому хотілося б перевірити всі шляхи взаємодії між підсистемами ПС. При цьому бажано хоч би протестувати всі ланцюжки виконання підсистем без повторного входження останніх. Якщо задана архітектура представляє ПС як малої системи з виділених підсистем, то число таких ланцюжків буде цілком осяжно.

*Тестування зовнішніх функцій*. Метою тестування є пошук розбіжностей між функціональною специфікацією і сукупністю програм ПС. Не дивлячись на те, що всі ці програми автономно вже відладжені, вказані розбіжності можуть бути, наприклад, із-за невідповідності внутрішніх специфікацій програм і їх модулів (на підставі яких проводилося автономне тестування) функціональної специфікації ПС. Як правило, тестування зовнішніх функцій проводиться так само, як і тестування модулів на першому кроці, тобто як чорного ящика.

*Тестування якості ПС.* Метою тестування є пошук порушень вимог якості, сформульованих в специфікації якості ПС. Це найбільш важкий і найменше вивчений вид тестування. Ясно лише, що далеко не кожен примітив якості ПС може бути випробуваний тестуванням. Завершеність ПС перевіряється вже при тестуванні зовнішніх функцій. На даному етапі тестування цього примітиву якості може бути продовжено, якщо потрібно одержати яку-небудь імовірнісну оцінку ступеня надійності ПС. Проте, методика такого тестування ще вимагає своєї розробки. Можуть тестуватися такі примітиви якості, як точність, стійкість, захищеність, тимчасова ефективність, в якійсь мірі − ефективність по пам'яті, ефективність по пристроях, розширюваність і, частково, незалежність від пристроїв. Кожний з цих видів тестування має свою специфіку і заслуговує окремого розгляду. Ми тут обмежимося лише їх переліком. Легкість застосування ПС оцінюється при тестуванні документації по застосуванню ПС.

*Тестування документації по застосуванню ПС.* Метою тестування є пошук неузгодженості документації по застосуванню і сукупністю програм ПС, а також виявлення незручностей, що виникають при застосуванні ПС. Цей етап безпосередньо передує підключенню користувача до завершення розробки ПС (тестуванню визначення вимог до ПС і атестацій ПС), тому вельми важливо розробникам спочатку самим скористатися ПС так, як це робитиме користувач. Всі тести на цьому етапі готуються виключно на підставі тільки документація по застосуванню ПС. Перш за все, повинні тестуватися можливості ПС як це робилося при тестуванні зовнішніх функцій, але тільки на підставі документації по застосуванню. Повинні бути протестовані всі неясні місця в документацію, а також всі приклади, використані в документації. Далі тестуються найбільш важкі випадки застосування ПС з метою виявити порушення вимог відносності легкості застосування ПС.

*Тестування визначення вимог до ПС.* Метою тестування є з'ясування, якою мірою ПС не відповідає пред'явленому визначенню вимог до нього. Особливість цього виду тестування полягає в тому, що його здійснює організація-покупець або організація-користувач ПС як один з шляхів подолання бар'єру між розробником і користувачем. Звичайно це тестування проводиться за допомогою контрольних завдань − типових завдань, для яких відомий результат рішення. У тих випадках, коли що розробляється ПС повинне придти на зміну іншої версії ПС, яка вирішує хоч би частину завдань того, що розробляється ПС, тестування проводиться шляхом рішення загальних задач за допомогою як старого, так і нового ПС (з подальшим зіставленням отриманих результатів). Іноді як форму такого тестування використовують *дослідну* експлуатацію ПС − обмежене застосування нового ПС з аналізом використання результатів в практичній діяльності. По суті, цей вид тестування багато в чому перекликається з випробуванням ПС при його атестації, але виконується до атестації, а іноді і замість атестації.

**Завдання:**

Обрати будь-яку задачу зі свого варіанту з л.р. 1 -6 або власну оригінальну задачу. Скласти блок-схему програми, її граф-модель (управляючий граф). Визначити всі різні прості шляхи руху від входу до виходу граф-моделі програми. Для кожного шляху скласти свій набір вхідних даних та отриманий вручну набір результатів обробки цих вхідних даних. За допомогою одержаних тестових прикладів продемонструвати працездатність програми.

***Контрольні питання:***

1. Що таке тестування програми?
2. Які методи тестування існують?
3. В чому полягає процес відладки програми?