Лекція 5
Ручне тестування коду програми

Зміст

Цільова настанова: Розглянуто проблематику ручного тестування коду програми, інспекції та наскрізні перегляди, інспекції початкового тексту тощо.

Ключові слова: тестування, якість програмного продукту, ручне тестування коду програми, інспекції та наскрізні перегляди, інспекції початкового тексту.

Вступ

Протягом багатьох років більшість програмістів були переконані в тому, що програми пишуться виключно для виконання їх на машині і не призначені для читання людиною, а єдиним способом тестування програми е ii виконання на ЕОМ. Ця думка стала змінюватися на початку 70-х років, а значною мірою — завдяки книзі Вейнберга «Психологія програмування для ЕОМ». Вейнберг довів, що програми повинні бути легкими для читання і що їх перегляд мае бути ефективним процесом виявлення помилок.

З цієї причини розглянемо процес тестування без застосування ЕОМ («ручного тестування»), що е темою цього розділу.

Експерименти показали, що методи ручного тестування досить ефективні з погляду знаходження помилок, так що один або декілька з них повинні використовуватися в кожному програмному проекті.

Описані тут методи призначені для періоду розробки, коли програма закодована, але тестування на ЕОМ ще не почалося. Аналогічні методи можуть бути отримані і застосовані на раніших етапах процесу створення програм (тобто в кінці кожного етапу проектування), але розгляд цього питання виходить за рамки даного посібника.

Слід зауважити, що через неформальну природу методів ручного тестування (неформальну з точки зору інших, формальніших методів, таких, як математичний доказ коректності програм) першою реакцією часто е скептицизм, відчуття того, що прості і неформальні методи не можуть бути корисними. Проте їх використання показало, що вони не «ве-

дуть убік». Швидше ці методи сприяють суттєвому збільшенню продуктивності і підвищенню надійності програми.

По-перше, вони зазвичай дозволяють раніше виявити помилки, зменшити вартість виправлення помилок і збільшити ймовірність того, що коригування проведено правильно.

По-друге, психологія програмістів, мабуть, змінюється, коли починається тестування на ЕОМ. Зростає внутрішня напруга, і з'являється тенденція «виправляти помилки так швидко, як тільки це можливо». В результаті програмісти допускають більше промахів при коригуванні помилок, вже знайдених під час тестування на ЕОМ, ніж при коригуванні помилок, знайдених на раніших етапах.

1. Інспекції та наскрізні перегляди

Інспекції вихідного тексту і наскрізні перегляди е основними методами ручного тестування. Оскільки ці два методи мають багато спільного, вони розглядаються тут спільно. Відмінності між ними обговорюються нижче.

Інспекції та наскрізні перегляди включають в себе читання або візуальну перевірку програми групою осіб. Ці методи розвинені з ідей Вейнберга. Обидва методи передбачають деяку підготовчу роботу.

Завершальним етапом с «обмін думками» зібрання, що проводиться учасниками перевірки. Мета такого зібрання — знаходження помилок, але не їх усунення (тобто тестування, а не налагодження).

Інспекції та наскрізні перегляди широко практикуються в даний час, але причини їх успіху досі ще недостатньо з’ясовані. Зауважимо, що даний процес виконується групою осіб (оптимально — три-чотири особи), лише один з яких е автором програми. Отже, програма, по суті, тестується не автором, а іншими людьми, які керуються викладеними вище принципами, зазвичай не ефективними при тестуванні власної програми.

Фактично, «інспекція» і «наскрізний перегляд» — просто нові назви старого методу «перевірки за столом», що полягає в тому, що програміст переглядає свою програму перед ii тестуванням. Однак вони набагато ефективніші знову-таки з тієї ж причини: у процесі бере участь не тільки автор програми, а й інші особи.

Результатом використання цих методів е нижча вартість налагодження (виправлення помилок), оскільки під час пошуку помилок зазвичай точно визначають їх природу.

Крім того, за допомогою даних методів виявляють групи помилок, що дозволяє надалі коригувати декілька помилок відразу. З іншого боку, при тестуванні на ЕОМ зазвичай виявляють тільки *симптоми* помилок (наприклад, програма не закінчилася або надрукувала безглуздий результат), а самі вони визначаються поодинці.

Експерименти по застосуванню цих методів показали, що з їх допомогою для типових програм можна знаходити від 30 до 70% помилок

логічного проектування та кодування (проте ці методи не ефективні при визначенні помилок проектування «високого рівня», наприклад зроблених у процесі аналізу вимог).

Так, експериментально встановлено, що при проведенні інспекцій та наскрізних переглядів визначаються в середньому 38% загального числа помилок у навчальних програмах.

При використанні інспекцій вихідного тексту в фірмі IBM ефективність виявлення помилок становить 80% (в даному випадку мається на увазі не 80% загального числа помилок, оскільки, як зазначалося раніше, загальне число помилок у програмі ніколи не відомо, а 80% всіх помилок, знайдених до моменту закінчення процесу тестування).

Звичайно, можна критикувати цю статистику в припущенні, що ручні методи тестування дозволяють знаходити тільки «легкі» помилки (ті, які можна просто знайти при тестуванні на ЕОМ), а важкі, непомітні або незвичайні помилки можна виявити тільки при тестуванні на машині. Однак проведені дослідження показали, що подібна критика с необгрунтованою.

Дослідники зробили також висновок про те, що при знаходженні певних типів помилок ручне тестування ефективніше, ніж тестування на ЕОМ, у той час як для інших типів помилок вірним е протилежне твердження. Мається на увазі, що інспекції, наскрізні перегляди і тестування, засноване на використанні ЕОМ, доповнюють один одного — ефективність виявлення помилок зменшиться, якщо той чи інший з цих підходів не буде використаний.

Нарешті, хоча ці методи дуже важливі при тестуванні нових програм, вони представляють не меншу цінність при тестуванні модифікованих програм. Досвід показав, що у разі модифікації існуючих програм вноситься більше число помилок (вимірюється числом помилок на знову написані оператори), ніж при написанні нової програми. Отже, модифіковані програми також повинні бути піддані тестуванню із застосуванням даних методів.

2. Інспекції початкового тексту

Інспекції початкового тексту являють собою набір процедур і прийомів виявлення помилок при вивченні (читанні) тексту групою фахівців. При розгляді інспекцій початкового тексту увагу буде зосереджено в основних процедурах, формах виконання тощо.

Група інспекторів включаючи до свого складу зазвичай чотири людини, одна з яких виконує функції голови. Голова повинен бути компетентним програмістом, але не автором програми; він не повинен бути знайомий з ii деталями.

В обов'язки голови входять підготовка матеріалів для засідань групи і складання графіка їх проведення, ведення засідань, реєстрація всіх знайдених помилок і прийняття заходів щодо їх подальшого виправлення.

Голову можна порівняти з інженером відділу технічного контролю. Членами групи є автор програми, проектувальник (якщо він не програміст) і фахівець з тестування.

Загальна процедура полягає в такому. Голова заздалегідь (наприклад, за кілька днів) роздає лістинг програми та проектну специфікацію іншим членам групи. Вони знайомляться з матеріалами перед засіданням. Інспекційне засідання розбивається на дві частини.

1. Програміста просять розповісти про логіку роботи програми. Під час бесіди виникають питання, які мають на меті виявлення помилки. Практика показала, що навіть тільки читання своєї програми слухачам видається ефективним методом виявлення помилок і багато помилок знаходить сам програміст, а не інші члени групи.

2. Програма аналізується за списком питань для виявлення історично сформованих загальних помилок програмування.

Голова є відповідальним за забезпечення результативності дискусії. Її учасники повинні зосередити свою увагу на знаходженні помилок, а не на їх коригуванні (коригування помилок виконується програмістом після інспекційного засідання).

Після закінчення засідання програмісту передається список знайдених помилок. Якщо список включаючи багато помилок або якщо ці помилки вимагають внесення значних змін, головою може бути прийнято рішення про проведення після коригування повторної інспекції програми. Список аналізується, і помилки розподіляються за категоріями, що дозволяє удосконалювати його з метою підвищення ефективності майбутніх інспекцій.

У більшості прикладів опису процесу інспектування стверджується, що під час інспекційного засідання помилки не повинні коригуватися. Однак існує й інша точка зору: замість того, щоб спочатку зосередитися на основних проблемах проектування, необхідно вирішити другорядні питання.

Дві або три людини, включаючи розробника програми, повинні внести очевидні виправлення в проект з тим, щоб згодом вирішити головні завдання. Під час обговорення найкращого способу внесення змін до проекту будь-хто з членів групи може помітити ще одну проблему.

Тепер групі доведеться розглядати дві проблеми стосовно до одних і тих же аспектів проектування, пояснення будуть повними і швидкими. Протягом декількох хвилин ціла область проекту може бути повністю досліджена, і будь-які проблеми стануть очевидними. Багато проблем, що виникали під час інспекції, можуть бути розв'язані під час оглядів блок-схем. Час і місце проведення інспекції повинні бути сплановані так, щоб уникнути будь-яких переривань інспекційного засідання. Його оптимальна тривалість лежить в межах від 90 до 120 хв. Оскільки це засідання є експериментом, що вимагає розумової напруги, збільшення його тривалості веде до зниження продуктивності.

Більшість інспекцій відбувається при швидкості, що дорівнює приблизно 150 операторам на годину. При цьому мається на увазі, що великі

програми повинні розглядатися за кілька інспекцій, кожна з яких може бути пов'язана з одним або декількома модулями або підпрограмами.

Для того щоб інспекція була ефективною, повинні бути встановлені відповідні відносини. Якщо програміст сприймає інспекцію як акт, спрямований особисто проти нього, і, отже, займає оборонну позицію, процес інспектування не буде ефективним. Програміст повинен підходити до інспекції з менш егоїстичної позиції; він повинен розглядати інспекцію в позитивному і конструктивному світлі.

Об'активно інспекція е процесом знаходження помилок в програмі і, таким чином, покращує якість ii роботи. 3 цієї причини, як правило, рекомендується результати інспекції вважати конфіденційними матеріалами, доступними тільки учасникам засідання. Зокрема, використання результатів інспекції керівництвом може завдати шкоди цілям цього процесу.

Процес інспектування, на додаток до свого основного призначення, що полягае в знаходженні помилок, виконує ще ряд корисних функцій. Крім того, що результати інспекції дозволяють програмісту побачити зроблені ним помилки і сприяють його навчанню на власних помилках, він зазвичай отримує можливість оцінити свій стиль програмування, вибір алгоритмів і методів тестування. Решта учасників також набувають досвіду, розглядаючи помилки і стиль програмування інших програмістів.

Нарешті, інспекція е способом раннього виявлення найбільш схильних до помилок частин програми, що дозволяє сконцентрувати увагу на цих частинах у процесі виконання тестування на ЕОМ.

3. Перелік запитань для виявлення помилок при інспекції

Важливою частиною процесу інспектування е перевірка програми на наявність загальних помилок за допомогою списку питань для виявлення помилок. Концентрація уваги в пропонованому списку на розгляді стилю, а не помилок представляється невдалою (питання типу «Чи с коментарі точними і інформативними?», «Чи розташовуються оператори THEN/ELSE і DO/END по одній вертикалі один під одним?», «У вас, мовляв, не ті кольори елементів екрану тощо). Список, що наведений у даному розділі, значною мірою е незалежним від мови. Це означає, що більшість помилок зустрічається в будь—якій мові програмування.

Помилки звернення до даних.

1. Чи існують звернення до змінних, значення яким не присвоєні або не ініціалізовані? Наявність змінних з не встановленими значеннями — програмна помилка, що зустрічається найчастіше, вона виникає при різних обставинах. Для кожного звернення до одиниці даних (наприклад, до змінної, елементу масиву, поля в структурі) спробуйте неформально «довести», що їй присвоено значення в точці, що перевіряється.
2. Чи не виходить значення кожного з індексів за межі, що визначені для відповідного вимірювання при всіх зверненнях до масиву?
3. Чи приймає кожен індекс цілі значення при всіх зверненнях до

масиву? Нецілі індекси не обов'язково е помилкою для всіх мов програмування, але представляють практичну небезпеку.

4. Для всіх звернень за допомогою покажчиків або змінних-посилань пам'ять, до якої проводиться звернення, вже розподілена? Наявність змінних- посилань являє собою помилку типу «підвішеного звернення». Вона виникає в ситуаціях, коли час життя покажчика більше, ніж час життя пам'яті, до якої проводиться звернення. Наприклад, до такого результату приводить ситуація, коли покажчик задає локальну змінну в тілі процедури, значення покажчика присвоюється вихідному параметру або глобальної змінної, процедура завершується (звільняючи адресну пам'ять), а програма потім намагається використовувати значення покажчика. Як і при пошуку помилок перших трьох типів, спробуйте неформально довести, що для кожного звернення, що використовує змінну-вказівник, адресна пам'ять існує.

5. Якщо одна і та ж область пам'яті має декілька псевдонімів (імен) із різними атрибутами, то чи мають значення даних в цій області коректні атрибути при зверненні по одному з цих псевдонімів? Помилки типу некоректних атрибутів у псевдонімів можуть виникнути при використанні атрибуту DEFINED або базованої пам'яті в PL/1, записів з варіантами в Паскалі або об'єднань (UNION) в мові С, накладання на одну і ту же ділянку пам'яті змінних з різними атрибутами (опція OVER в Clarion). Як приклад можна навести програму на мові С, що містить дійсну змінну *A* і цілу змінну *B*. Обидві величини розміщені на одному і тому ж місці пам'яті (тобто поміщені в одне й теж об'єднання). Якщо програма записує величину *A,* а звертається до змінної *B,* то, ймовірно, відбудеться помилка, оскільки машина буде використовувати бітове представлення числа з плаваючою точкою в даній області пам'яті як ціле.

6. Якщо використовуються покажчики або змінні-посилання, то чи мають адресну пам'ять атрибути, передбачувані компілятором? Прикладом невідповідності атрибутів може служити випадок, коли покажчик, за яким базується структура даних, має як значення адресу іншої структури-

7. Якщо до структури даних звертаються з декількох процедур або підпрограм, то чи визначена ця структура однаково в кожній процедурі?

8. Чи не перевищені межі рядка при індексації в ньому?

9. Чи існують які-небудь інші помилки в операціях з індексацією або при зверненні до масивів за індексом?

Помилки опису даних.

1. Чи всі змінні описані явно? Відсутність явного опису не обов'язково е помилкою, але служить спільним джерелом занепокоєння. Так, якщо в підпрограмі на Фортрані використовується елемент масиву і відсутній його опис (наприклад, в операторі DIMENSION), то звернення до масиву (наприклад, *= A (1)),* буде інтерпретуватися як виклик функції. Останне призведе до того, що машина буде намагатися обробити масив як програму. Якщо відсутній явний опис змінної у внутрішній процедурі або блоці, чи слід розуміти це так, що опис даної змінної

співпадає з описом в зовнішньому блоці?

2. Якщо не всі атрибути змінної явно присутні в описі, то чи зрозуміла їх відсутність? Наприклад, відсутність атрибутів, що вважається загальноприйнятим в PL/1 або Clarion, часто е джерелом несподіваних ускладнень.

3. Якщо початкові значення присвоюються змінним в операторах опису, то чи правильно ініціалізуються ці значення? У багатьох мовах програмування привласнення початкових значень масивам і рядкам представляється досить складним і, отже, е можливим джерелом помилок.

4. Чи правильно для кожної змінної визначені довжина, тип і клас пам'яті (наприклад, STATIC, AUTOMATIC, AUTO тощо)?

5. Чи узгоджується ініціалізація змінної з ii типом пам'яті? Наприклад, якщо в PL/1 (або Clarion) описується ініціалізація величини і початкове значення необхідно встановлювати кожен раз при виклику процедури, то для цієї змінної має бути визначений клас пам'яті AUTOMATIC, а не STATIC.

6. Чи е змінні з подібними іменами (наприклад, NUMBER і NAMBERS)? Наявність подібних імен не обов’язково е помилкою, але служить ознакою того, що імена можуть бути переплутані де-небудь усередині програми.

Помилки обчислень.

1. Чи е обчислення, що використовують змінні неприпустимих типів даних (наприклад, неарифметичних)?

2. Чи існують обчислення, що використовують дані різного виду? Наприклад, додавання змінної з плаваючою точкою і цілої змінної. Такі випадки не обов’язково е помилковими, але вони повинні бути ретельно перевірені для забезпечення гарантії того, що правила перетворення, що прийняті в мові, зрозумілі. Це особливо важливо для мов зі складними правилами перетворення (наприклад, для PL/1, C). Наприклад, наступний фрагмент програми на мові C, спроба привласнити змінній з плаваючою крапкою значення 1/2 дасть значення 0 (тому що 1 і 2 мають цілий тип).

3. Чи існують обчислення, що використовують змінні, що мають однаковий тип даних, але різну довжину? Таке питання справедливе для PL/1, і виникло воно з цієї мови. Наприклад, у мові C присутня множина даних одного типу, але вони мають різну довжину. Наприклад, це цілі типи (CHAR, INT, LONG, SHORT, INT64), дійсні типи (FLOAT, DOUBLE, LONG DOUBLE) тощо. До того ж, ці типи можуть бути, як знаковими (SIGNED), так і беззнаковими (UNSIGNED).

4. Чи має результуюча змінна оператора присвоювання атрибути, що описують *ii* з меншою довжиною, ніж в атрибутах виразу в правій частині?

5. Чи можливі переповнення або втрата результату під час обчислення виразу? Це означає, що кінцевий результат може здаватися пра-

вильним, але проміжний результат може бути занадто великим або занадто малим для машинного представлення даних.

6. Чи можливо, щоб дільник в операторі ділення дорівнював нулю?

7. Якщо величини представлені у машині в двійковій формі, чи будуть якісь результати неточними? Так, *10*0,1* рідко буде дорівнювати *1,0* в двійковій машині.

8. Чи може значення змінної виходити за межі встановленого для неї діапазону? Наприклад, для операторів, які присвоюють значення змінної PROB (що має сенс ймовірності якої-небудь події), може бути проведена перевірка, чи буде отримане значення завжди позитивним і не перевищує 1,0.

9. Чи вірні припущення про порядок оцінки і прямування операторів для виразів, що містять більш ніж один оператор?

10. Чи зустрічається невірне використання цілої арифметики, особливо ділення? Наприклад, якщо *I* ціла величина, то вираз *I—2*1/2* залежить від того, е значення / парним або непарним, і від того, яка дія — множення або ділення — виконується першою.

Помилки при порівняннях.

1. Чи порівнюються в програмі величини, що мають несумісні типи даних (наприклад, рядок символів з адресою)?

2. Чи порівнюються величини різних типів або величини різної довжини? Якщо так, то перевірте, чи правильно інтерпретуються (зрозумілі) правила перетворення.

3. Чи коректні оператори порівняння? Деякі програмісти часто плутають такі відносини як найбільший, найменший, більше чому, не менше ніж, менше або дорівнює.

4. Чи кожний булевський вираз сформульовано так, як це передбачалося? Програмісти часто роблять помилки при написанні логічних виразів, що містять операції «І», «АБО», «НІ».

5. Чи е операнди булєвских виразів булєвськими? Чи існують помилкові об'єднання порівнянь і булєвских виразів? Вони представляють інший клас помилок, який зустрічається дуже часто. Приклади кількох типових помилок наведені нижче.

Якщо величина *L* визначена так, що лежить в інтервалі між 2 і 10, то вираз *2 <L 310 в* невірним. Замість нього має бути написано вираз *(2 <L) & (L <10).*

Якщо ж величина *L* визначена як більша, ніж J або *У,* то вираз *L > | У* с невірним. Він має бути записаний у вигляді LX J2 | §L> *У).*

При порівнянні трьох чисел на рівність вираз /N *(А — В С)* означає зовсім інше. Наприклад, у мові С станеться привласнення змінним *А* і *В* значення змінної *С.* А умова буде істинною, якщо це значення ненульове.

У разі необхідності перевірити математичне відношення = *У — Z* правильним буде вираз *(J = У) & (У Z).* Також в мові С слід розрізняти булєвські і бітові оператори. Наприклад, якщо *А* — 7 і *В 2,* то умова *IF (А && В)* буде істинною, а /N *(А & В)* — помилковою.

6. Чи порівнюються в програмі числа в десяткових формах подання з числами з плаваючою комою, які представлені в машині в двійковій формі? Це с іноді джерелом помилок через неточну рівність чисел у двійковій і десятковій формах подання.

7. Чи вірні припущення про порядок оцінки і проходженні операторів для виразів, що містять більше одного булєвського оператора? Іншими словами, якщо задано вираз *(А ——2)&(В ——2)|(С——3),* чи зрозуміло, яка з операцій виконується першою: И або ИЛИ?

8. Чи впливає на результат виконання програми спосіб, яким конкретний компілятор виконує булеві вирази? Наприклад, оператор *IF (X — 0) &* ((Г/Jj> *Z) с* прийнятним для деяких компіляторів (тобто компіляторів, які закінчують перевірку, як тільки один з виразів оператора «/f» виявиться хибним), але призведе до ділення на 0 при використанні інших компіляторів.

Помилки в передачах управління.

1. Чи буде кожен цикл, зрештою, завершено? Придумайте неформальний доказ або аргументи, що підтверджують їх завершення.

2. Чи будуть програма, модуль або підпрограма, в кінцевому рахунку, завершені?

3. Чи можливо, що через вхідні умови цикл ніколи не зможе виконуватися? Якщо це так, то чи с це необачністю? Наприклад, що відбудеться для циклів, що починаються операторами:

DO WHILE
(ABC) DO
I—X TO Z

якщо початкове значення *ABC* — хиба або якщо більше Z?

4. Чи існують які—небудь помилки «відхилення від норми» ? Наприклад, занадто велике або занадто мале число ітерацій.

5. Якщо мова програмування містить поняття групи операторів (наприклад, DO-групи в PL/1, обмежені операторами *DO-END, BEGIN... END* в Паскалі), то чи е явний оператор *END* для кожної групи і чи відповідають оператори *END* своїм групам?

6. Чи існують рішення, що припускаються за умовчанням? Наприклад, нехай очікується, що вхідний параметр приймає значення 1, 2 або 3. Чи логічно тоді припустити, що він повинен бути рівний 3, якщо він не дорівнює 1 або 2? Наприклад, розглянемо програму на мові С:

```
switch(X) {
case 1. printf(“1!!!”); break;
case 2. printf(“2!!!”), break,
default. printf(“3!!!”);
```

Коли це так, то чи е припущення правильним?

Помилки інтерфейсу.

1. Чи дорівнює число параметрів, що одержуються модулем, числу аргументів, переданих кожним з викликаючих модулів? Чи правильний порядок їх послідовності?

2. Чи збігаються атрибути (наприклад, тип і розмір) кожного параметра з атрибутами відповідного йому аргументу?

3. Чи збігаються одиниці виміру кожного параметра з одиницями вимірювання відповідних аргументів? Наприклад, чи немає випадків, коли значення параметрів виражено в градусах, а аргументу — в радіанах?

4. Чи дорівнює число аргументів, переданих з розглянутого модуля іншому модулю, числу параметрів, очікуваних в викликаючому модулі?

5. Чи відповідають атрибути кожного аргументу, переданого іншому модулю, атрибутам відповідного параметра в розглянутому модулі?

6. Чи збігаються одиниці виміру кожного аргументу, переданого іншому модулю, з одиницями вимірювання відповідної дії в розглянутому модулі?

7. Якщо викликаються вбудовані функції, чи правильно задані кількість, атрибути та порядок проходження аргументів?

8. Чи не змінює підпрограма параметр, який повинен використовуватися тільки як вхідна величина?

9. Якщо е глобальні змінні (наприклад, змінні в PL/1, Clarion з атрибутом EXTERNAL або в С з атрибутом EXTERN), чи мають вони однакові визначення та атрибути у всіх модулях, які до них звертаються?

Помилки вводу-виводу.

1. Чи е правильними атрибути файлів, описаних явно?

2. Чи с правильними атрибути оператора OPEN?

3. Чи узгоджується специфікація формату з інформацією в операторах вводу-виводу?

4. Чи мають одинакові розміри запису та розмір області пам'яті для введення-виведення? Це може бути важливо при блочному введенні-виведенні (функціі BLOCKREAD і BLOCKWRITE в Паскалі, FREAD і FWRITE в Сі).

5. Чи всі файли відкриті перед їх використанням?

6. Чи правильно виявляються і трактуються ознаки кінця файлу?

7. Чи правильно трактуються помилкові стани вводу-виводу?

8. Чи існують смислові або граматичні помилки в тексті, виведеному програмою на друк або екран дисплея?

Інші види контролю.

1. Якщо компілятор видає таблицю перехресних посилань ідентифікаторів, перевірте величини, на які в цьому списку немає посилань або е тільки одне посилання.

2. Якщо компілятор видає список атрибутів, перевірте атрибути кожної величини для забезпечення гарантії того, що в програмі немає ніяких несподіваних і відсутніх атрибутів.

3. Якщо програма відтрансльована успішно, але компілятор видає одне або декілька «попереджень» або «інформаційних» повідомлень, уважно перевірте кожне з них. Попередження свідчить про «підозри»

компілятора щодо правильності ваших дій. Всі ці «підозри» повинні бути розглянуті. В інформаційних повідомленнях можуть перераховуватися неописані змінні або конструкції мови, які перешкоджають оптимізації коду.

4. Чи є програма (або модуль) досить стійкою? Іншими словами, чи перевіряє вона правильність своїх вхідних даних?

5. Чи не пропущена в програмі якась функція?

- *Наскрізні перегляди*

Наскрізний перегляд, як і інспекції, являє собою набір процедур і способів виявлення, що здійснюються групою осіб, які переглядають текст програми. Такий перегляд має багато спільного з процесом інспектування, але їх процедури дещо відрізняються, і, крім того, тут використовуються інші методи виявлення помилок.

Подібно інспекції, наскрізний перегляд проводиться як безперервне засідання, що триває одну або дві години. Група з виконання наскрізного перегляду складається з 3-5 осіб. До її складу входять голова, функції якого подібні до функцій голови в групі інспектування, секретар, який записує всі знайдені помилки, і фахівець з тестування.

Думки про те, хто повинен бути четвертим і п'ятим членами групи, розходяться. Звичайно, одним з них повинен бути програміст. Щодо п'ятого учасника є такі припущення:

1) висококваліфікований програміст;
2) експерт з мови програмування;
3) початківець, на точку зору якого не впливає попередній досвід;
4) людина, яка буде експлуатувати програму;
5) учасник якого-небудь іншого проекту;
6) хто-небудь з тієї ж групи програмістів, що і автор програми.

Початкова процедура при наскрізному перегляді така ж, як і при інспекції: учасникам заздалегідь, за кілька днів до засідання, роздаються матеріали, що дозволяють їм ознайомитися з програмою. Однак процедура засідання відрізняється від процедури інспекційного засідання. Замість того, щоб просто читати текст програми або використовувати список помилок, учасники засідання «виконують роль обчислювальної машини».

Особа, яка призначена тестуючим, пропонує присутнім невелике число написаних на папері тестів, що представляють собою набори вхідних даних (і очікуваних вихідних даних) для програм або модуля. Під час засідання кожен тест виконується подумки. Це означає, що тестові дані піддаються обробці відповідно до логіки програми. Стан програми (тобто значення змінних) відстежується на папері або дошці.

Звичайно, число тестів має бути невеликим і вони повинні бути простими за своєю природою, тому що швидкість виконання програми людиною на багато порядків менше, ніж у машини. Отже, тести самі по собі не відіграють критичної ролі,

Як і при інспекції, думка учасників е вирішальним фактором. Зауваження повинні бути адресовані скоріше вони служать засобом для початкового розуміння програми і основою для питань програмісту про логіку проектування та прийнятих припущень. У більшості наскрізних переглядів при виконанні самих тестів знаходять менш помилок, ніж при опитуванні програміста. Іншими словами, помилки не розглядаються як слабкість людини, яка їх допустила. Вони свідчать про складність процесу створення програм і є результатом все ще примітивної природи існуючих методів програмування.

Наскрізні перегляди повинні протікати так само, як і описаний раніше прогрес інспектування. Побічні ефекти, одержувані під час виконання цього процесу (встановлення схильних до помилок частин програми і навчання на основі аналізу помилок, стилю і методів), також характерні і для процесу наскрізних переглядів.

- *Оцінка за допомогою перегляду*

Розглянемо останній ручний процес огляду програми, який також не пов'язаний з ii тестуванням, тобто метою його не е знаходження помилок. Однак опис цього процесу наводиться тут тому, що він має відношення до ідеї читання тексту.

Оцінка за допомогою перегляду е методом оцінки анонімної програми в термінах ii загальної якості, ремонтопридатності, розширюваності, простоти експлуатації та ясності.

Мета даного методу — забезпечити програміста засобами самооцінки. Вибирається програміст, який повинен виконувати обов’язки адміністратора процесу. Адміністратор, в свою чергу, відбирає приблизно 6-20 учасників (6 — це є мінімальне число для збереження анонімності). Передбачається, що учасники повинні бути одного профілю (наприклад, в одну групу не слід об'єднувати програмістів, що використовують Паскаль, і системних програмістів, які пишуть на Асемблері). Кожного учасника просять представити для розгляду дві свої програми — найкращу (з його точки зору) і низької якості.

Відібрані програми випадковим чином розподіляються між учасниками. Їм дається на розгляд по чотири програми. дві з них є «найкращими», а дві — «найгіршими», але рецензенту не повідомляють про те, яка програма до якої групи належить.

Кожен учасник витрачає на перегляд однієї програми 30 хв і заповнює анкету для ii оцінки. Після перегляду всіх чотирьох програм оцінюється їх відносна якість. В анкеті для оцінки перевіряючому пропонується оцінити програму за семибальною шкалою (1 означає певно «так», 7 певно «ні») при відповіді, наприклад, на такі питання:

1. Чи легко було зрозуміти програму?
2. Чи виявилися результати проектування високого рівня очевидними і прийнятними?
3. Чи виявилися результати проектування низького рівня очевидними і прийнятними?

4. Чи легко для вас модифікувати цю програму?
5. Випробовували 6 ви почуття задоволення, написавши таку програму?

Перевіряючого просять також дати загальний коментар і рекомендації щодо поліпшення програми. Після перегляду кожному учаснику передають анонімну анкету з оцінкою двох його програм. Учасники отримують статистичне зведення, яке містить загальну і детальну класифікацію їх власних програм в порівнянні з повним набором програм, і аналіз того, наскільки оцінки чужих програм збігаються з оцінками тих же самих програм, даними іншими перевіряючими.

Мета такого перегляду — дати можливість програмістам самим оцінити свою кваліфікацію. Цей спосіб представляється корисним як для промислового, так і для навчального застосування.

Контрольні питання

1. Інспекції та наскрізні перегляди.
2. Який набір процедур являють собою інспекції початкового тексту?
3. *Які* існують помилки звернення до даних?
4. Які існують помилки опису даних?
5. Помилки обчислень.
6. Помилки при порівняннях.
7. Помилки в передачах управління.
8. Помилки інтерфейсу.
9. Помилки вводу-виводу.
10. Оцінка огляду програми за допомогою перегляду.