Лекція 6.

Регресійне тестування: цілі і завдання, умови застосування, класифікація тестів і методів відбору

Зміст

Цільова настанова: Розглядаються цілі, завдання та види регресійного тестування. Перераховуються необхідні і достатні умови застосування методів вибіркового регресійного тестування. Дається класифікація методів вибіркового регресійного тестування і самих тестів при відборі. Розглядаються можливості повторного використання тестів.

Ключові слова: регресійні тестування, цілі регресійного тестування, завдання регресійного тестування, види регресійного тестування, регресійний дефект, вибіркове регресійне тестування, коригуючий супровід, адаптивне супровід, кероване регресійне тестування, вибіркове регресійне тестування, відбір тестів, повторне використання тестів, новий тест, повторно використовуваний тест, повторно запускається тест, застарілий тест.

12.1. Цілі і завдання регресійного тестування

При коригування програми необхідно гарантувати збереження якості. Для цього використовується регресійне тестірованіе- дорога, але необхідна діяльність в рамках етапу супроводу, спрямована на перевірку коректності зміненої програми. У відповідності зі стандартним визначенням, регресійні тестування - це вибіркове тестування, що дозволяє переконатися, що зміни не викликали небажаних побічних ефектів, або що змінена система як і раніше відповідає вимогам.

Головним завданням етапу супроводу є реалізація систематичного процесу обробки змін в коді. Після кожної модифікації програми необхідно упевнитися, що на функціональність програми не вплинув модифікований код.

Якщо такий вплив виявлено, говорять про регрессионном дефекті. Для регресійного тестування функціональних можливостей, зміна яких не пла-лося раніше, використовуються раніше розроблені тести. Одна з цілей регресійного тестування полягає в тому, щоб, відповідно до використовуваним критерієм покриття коду (наприклад, критерієм покриття потоку операторів або потоку даних), гарантувати той же рівень покриття, що і при повному повторному тестуванні програми. Для цього необхідно запускати тести, які стосуються зміненим областям коду або функціональними можливостями.

Нехай T = {tj, t2, ..., tN} - безліч з N тестів, що використовується при первинній розробці програми P, а T 'T - підмножина регресійних тестів для тестування нової версії програми P'. Інформація про покриття коду, яке забезпечується T ', дозволяє вказати блоки P', що вимагають додаткового тестування, для чого може знадобитися повторний запуск деяких тестів з безлічі T T ', або навіть створення T "- набору нових тестів для P' - і оновлення T.

Інша мета регресійного тестування полягає в тому, щоб упевнитися, що програма функціонує у відповідності зі своєю специфікацією, і що зміни не призвели до внесення нових помилок в раніше протестований код. Ця мета завжди може бути досягнута повторним виконанням всіх тестів регресійного набору, але більш перспективно відсівати тести, на яких вихідні дані модифікованої і старій програми не можуть різнитися. Результати порівняння вибіркових методів і методу повторного прогону всіх тестів наведені в табл. 11.1.

Табл. 11.1. Вибіркове регресійне тестування і повторний прогін всіх тестів

Повторний прогін всіх тестів	Вибіркове регресійне тестування
Простий в реалізації	Вимагає додаткових витрат при впровадженні
Дорогий і неефективний	Здатне зменшувати витрати за рахунок виключення зайвих тестів
Виявляє всі помилки, які були б знайдені при вихідному тестуванні	Може призводити до пропуску помилок

Завдання відбору тестів з набору T для заданої програми P і зміненої версії цієї програми P 'полягає у виборі підмножини T' ідеальне T Для повторного запуску на зміненій програмі P ', де T'ідеальное = {t T | P' (t) P (t)}.

Так як вихідні дані P і P 'для тестів з безлічі T T'ідеальное свідомо однакові, немає необхідності виконувати жоден з цих тестів на P'. У загальному випадку, за відсутності динамічної інформації про виконання P і P 'не існує методики обчислення безлічі T'ідеальное для довільних множин P, P' і T. Це випливає з відсутності спільного рішення проблеми зупинки, що складається в неможливості створення в загальному випадку алгоритму, дає відповідь на питання, завершується чи коли-небудь довільна програма P для заданих значень вхідних даних. На практиці створення T 'ідеальне можливо тільки шляхом виконання на інструментований версії P' кожного регресійного тесту, чого і хочеться уникнути.

Реалістичний варіант вирішення завдання вибіркового регресійного тестування полягає в отриманні корисної інформації за результатами виконання P і об'єднання цієї інформації з даними статичного аналізу для отримання безлічі T 'реальне у вигляді апроксимації T' ідеальне. Цей підхід застосовується у всіх відомих вибіркових методах регресійного тестування, заснованих на аналізі коду. Безліч T'реа-льно має включати всі тести з T, що активують змінений код, і не включати ніяких інших тестів, тобто тест t T входить в T 'реальне тоді і тільки тоді, коли t задіє код P в точці, де в P 'код був вилучений або змінений, або де був доданий новий код.

Якщо деякий тест t задіє в P той же код, що і в P ', вихідні дані P і P' для t відрізнятися не будуть. З цього випливає, що якщо P (t) P '(t), t повинен задіяти деякий код, змінений в P' по отноше-нию до P, тобто має виконуватися відношення t T 'реальне. З іншого боку, оскільки не кожне виконання зміненого коду відбивається на вихідних значеннях тесту, можуть існувати деякі такі t T 'реальне, що P (t) = P' (t). Таким чином, T 'реальне містить T' ідеальне цілком і може використовуватися в якості його альтернативи без шкоди для якості тестованого програмного продукту.

Важливим завданням регресійного тестування є також зменшення вартості і скорочення часу виконання тестів.

Наприклад, якщо код, що покриваються тестом, не змінився з попередньої версії, отже, повторне виконання даного тесту не потрібно. Якщо ж код, що покриваються тестом, змінився; отже, необхідний його повторний запуск.

12.2. Види регресійного тестування

Оскільки регресійне тестування є повторне проведення циклу звичайного тестування, види регресійного тестування збігаються з видами звичайного тестування. Можна говорити, наприклад, про модульному регрессионном тестуванні або про функціональне регрессионном тестуванні.

Інший спосіб класифікації видів регресійного тестування пов'язує їх з типами супроводу, які, в свою чергу, визначаються типами модифікацій. Виділяють три типи супроводу:

- Коригувальна супровід, зване зазвичай виправленням помилок, виконується у відповідь на виявлення помилки, яка потребує зміни специфікації вимог. При коректує супроводі проводиться діагностика і коригування дефектів в програмному забезпеченні з метою підтримки системи в працездатному стані.
- Адаптивне супровід здійснюється у відповідь на вимоги зміни даних або середовища виконання. Воно застосовується, коли існуюча система поліпшується або розширюється, а специфікація вимог змінюється з метою реалізації нових функцій.
- Вдосконалять (прогресивне) супровід включає будь-яку обробку з метою підвищення ефективності роботи системи або ефективності її супроводу.

В процесі адаптивного або вдосконалять супроводу зазвичай вводяться нові модулі. Щоб відобразити ту чи іншу удосконалення або адаптацію, змінюється специфікація системи. При коректує супроводі, як правило, специфікація не змінюється, і нові модулі не вводяться. Модифікація програми на фазі розробки подібна модифікації при коректує супроводі, так як через виявлення помилки навряд чи потрібно міняти специфікацію програми. За винятком рідкісних моментів великих змін, на фазі супроводу зміни системи зазвичай невеликі і виробляються з метою усунення проблем або поступового розширення функціональних можливостей.

Відповідно, визначають два типи регресивного тестування: прогресивне і коригуючий:

- Прогресивне регресійне тестування передбачає модифікацію технічного завдання. У більшості випадків при цьому до системи програмного забезпечення додаються нові модулі.
- При коректує регрессионном тестуванні технічне завдання не змінюється. Модифікуються тільки деякі оператори програми і, можливо, конструкторські рішення.

Прогресивне регресійне тестування зазвичай виконується після адаптивного або вдосконалять супроводу, тоді як коригувальну регресійне тестування виконується під час тестування в циклі розробки і після коригуючого супроводу, тобто після того, як над програмним забезпеченням були виконані деякі коригувальні дії. Взагалі кажучи, коригуючий регресійне тестування повинно бути простішим, ніж прогресивне регресійне тестування, оскільки допускає повторне використання більшої кількості тестів.

Підхід до відбору регресійних тестів може бути активним або консервативним. Активний підхід на перше місце ставить зменшення обсягу регресійного тестування і нехтує ризиком пропустити дефекти. Активний підхід застосовується для тестування систем з високою вихідною надійністю, а також у випадках, коли ефект змін невеликий. Консервативний підхід вимагає відбору всіх тестів, які з ненуліовою ймовірністю можуть виявляти дефекти. Цей підхід дозволяє виявляти більшу кількість помилок, але призводить до створення більш великих наборів регресійних тестів.

12.3. Кероване регресійне тестування

Протягом життєвого циклу програми період супроводу довгих-ся довго. Коли змінена програма тестується набором тестів T, ми зберігаємо без змін по відношенню до тестування вихідної програми P всі фактори, які могли б впливати на висновок програми. Тому атрибути конфігурації, в якій програма тестірован останній раз (наприклад, план тестування, тести tj і покриваються елементи MT (P, C, tj)), підлягають управлінню конфігурацією.

Практика тестування зміненої версії програми P 'в тих же умовах, в яких тестувалася вихідна програма P, називається керованим регресійний тестуванням. При некерованому регрессионном тестуванні деякі властивості методів

регресійного тестування можуть змінюватися, наприклад, безпечний метод відбору тестів може перестати бути безпечним. У свою чергу, для забезпечення керованості регресійного тестування необхідно виконання ряду умов:

1. Як при модульному, так і при інтеграційному регрессионном тестуванні в якості модулів, що викликаються тестованим модулем безпосередньо або побічно, повинні використовуватися реальні модулі системи. Це легко здійснити, оскільки на етапі регресійного тестування всі модулі присутні в завершеному вигляді.

2. Інформація про зміни коректна. Інформація про зміни вказує на змінені модулі та розділи специфікації вимог, не маючи на увазі при цьому коректність самих змін. Крім того, при зміні специфікації вимог необхідно посилене регресійне тестування змінилися функцій цієї специфікації, а також всіх функцій, які могли бути порушені через необережність. Єдиним випадком, коли ми змушені поло-житися на правильність зміненого технічного завдання, є зміна технічного завдання для всієї системи або для модуля верхнього (в графі викликів) рівня, за умови, що крім технічного завдання, не існує ніякої додаткової документації і / або який -або іншої інформації, по якій можна було б судити про помилку в технічному завданні.

3. У програмі немає помилок, крім тих, які могли виникнути через її зміни.

4. Тести, що застосовувалися для тестування попередніх версій програмного продукту, доступні, при цьому протокол прогону тестів складається з вхідних даних, вихідних даних і траєкторії. Траєкторія являє собою шлях в керу-ючому графові програми, проходіння якого викликається використанням де-якого набору вхідних даних. Її можна застосовувати для оцінки структурного покриття, що забезпечується набором тестів.

5. Для проведення регресійного тестування з використанням су-суспільством набору тестів необхідно зберігати інформацію про результати виконання тестів на попередніх етапах тестування.

Припустимо, що ніякі оператори програми, крім тих, чиє по-ведення залежить від змін, не можуть несприятливо впливати на програму. Навіть за такої умови існують деякі ситуації, які потребують особливої уваги, наприклад, проблема витоку пам'яті і їй подібні. Ситуації такого роду в різних системах програмування обробляються по-різному. Наприклад, мова Java сам по собі включає систему управління пам'яттю. Якщо ж система не контролює распреде-ня пам'яті автоматично, ми повинні вважати, що всі оператори ра-боти з пам'яттю також володіють поведінкою, що залежать від змін.

Проблема мов типу C і C ++, які допускають довільні арифметичні операції над покажчиками, полягає в тому, що покажчики можуть порушувати межі областей пам'яті, на які вони вказують. Це означає, що змінні можуть оброблятися способами, які не піддаються аналізу на рівні вихідного коду. Щоб врахувати такі на-рушення кордонів пам'яті, висуваються наступні гіпотези:

• Гіпотеза 1 (Чітко визначена пам'ять). Кожен сегмент пам'яті, до якого звертається система програмного забезпечення, соот-ветствует деякої символічно певної змінної.

• Гіпотеза 2 (Строго обмежений покажчик). Кожна змінна або вираз, що використовується як покажчик, має посилатися на деяку базову змінну і обмежуватися використанням сегмента пам'яті, що визначається цієї змінної.

Щоб гарантувати покриття всіх залежних від змін кому-тами, для яких, можна показати, що вони будуть зачіпатися існуючими тестами, досить одного тесту для кожного з таких компонентов. Безліч тестів досить великого роз-міру (як правило, сценарних), може сприяти виявленню помилок, викликаних порушеннями умов керованого регресійного тестування. Існують і організаційні умови проведення регресійного тестування. Це ресурс (час), необхідний тестовому аналітику для ознайомлення зі специфікацією вимог системи, її архітектурою і, можливо, самим кодом.

11.4. Обгрунтування коректності методу відбору тестів

Перерахуємо деякі особливості реалізації регресійного тестування:

1. Деякі ділянки коду програми не отримують управління при виконанні деяких тестів. Якщо ділянку коду реалізує вимогу, але змінений фрагмент коду не отримує управління при виконанні тесту, то він і не може впливати на значення вихідних даних програми при виконанні даного тесту.

2. Навіть якщо ділянка коду, який реалізує вимога, отримує управління при виконанні тесту, це далеко не завжди відбивається на вихідних даних програми при виконанні даного тесту. Дійсно, якщо змінюється перший блок програми, наприклад, шляхом додавання ініціалізації змінної, всі шляхи в програмі також змінюються, і, як наслідок, вимагають повторного тестірова-ня. Однак може так статися, що тільки на невеликому підмножині шляхів дійсно використовується ця инициализированва змінна.

3. Не кожен тест tk T, перевіряючий код, що знаходиться на одному шляху зі зміненим кодом, обов'язково покриває цей змінений код.

4. Код, що знаходиться на одному шляху зі зміненим кодом, може не впливати на значення вихідних даних змінених модулів програми.

5. Не завжди кожен оператор програми впливає на кожен елемент її вихідних даних.

Припустимо, що зміни в програмі обмежуються одним оператором. Якщо при виконанні будь-якого тесту на вихідній програмі цей оператор ніколи не отримує управління, можна з упевненістю сказати, що він не отримає управління і в ході виконання тесту на нову програму, а результати тестування нової і старої програм будуть збігатися. Отже, немає необхідності виконувати цей тест на нову програму.

Зазначений метод легко можна узагальнити для випадку декількох змін: якщо тест не задіє жодного зміненого оператора, і його вхідні дані не змінилися, код, що виконується їм у зміненій програмі, буде в точності таким же, як в первісній версії. Такий тест не виявляє відмінностей між двома версіями системи; отже, немає необхідності проганяти його повторно. Якщо тест не зачіпає жодного оператора виведення, поведінка якого залежить від змінених операторів, це означає, що, незважаючи на зміни в програмі, всі оператори, які

отримують управління при виконанні цього тесту, не змінять висновок системи по відношенню до попередньої версії. Таким чином, немає необхідності повторно проганяти і тести такого роду.

Отже, необхідно орієнтуватися на вибір тільки тих тестів, які покривають змінений код, що впливає, в свою чергу, на висновок програми. Такий підхід гарантує, що будуть обрані тільки тести, які виявляють зміни, і метод буде, як кажуть, точним.

12.5. Класифікація тестів при відборі

Створення наборів регресійних тестів рекомендується починати з безлічі вихідних тестів. При заданому критерії регресійного тестування всі вихідні тести t (tkT) поділяються на три підмножини:

1. Безліч тестів, придатних для повторного використання. Це тести, які вже запускалися і придатні до використання, але зачіпають лише покриваються елементи програми, що не зазнали змін. При повторному виконанні вихідні дані таких тестів співпадуть з вихідними даними, отриманими на вихідній програмі. Отже, такі тести не вимагають перезапуску.
2. Безліч тестів, які потребують повторного запуску. До них відно-сятся тести, які вже запускалися, але вимагають перезапуску, оскільки зачіпають, по крайней мере, один змінений покриваються елемент, що підлягає повторному тестуванню. При повторному виконанні такі тести можуть давати результат, відмінний від результату, показаного на вихідній програмі. Безліч тестів, які потребують повторного запуску, забезпечує хороше покриття структурних елементів навіть при наявності нових функціональних можливостей.
3. Безліч застарілих тестів. Це тести, паче не застосовні до зміненої програмою і непридатні для подальшого тестування, оскільки вони зачіпають лише покриваються елементи, які були видалені при зміні програми. Їх можна видалити з набору регресійних тестів.
4. Нові тести, які ще не запускалися і можуть бути використані для тестування.

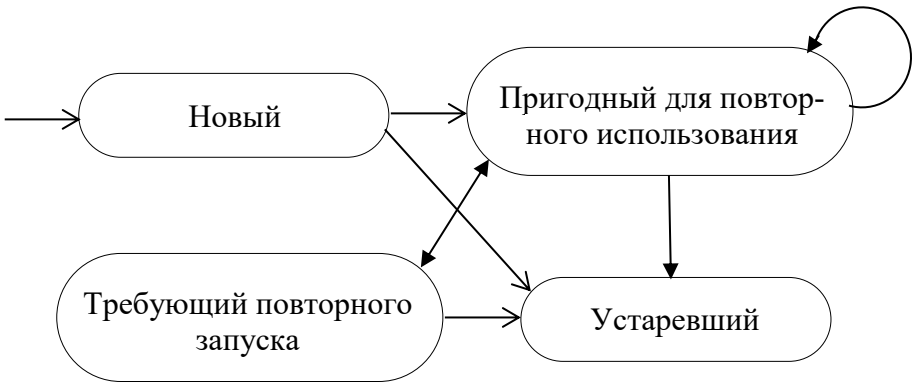

Рис. 12.2. Життєвий цикл тесту

Рис. 12.2 дає уявлення про життєвий цикл тесту. Відразу після створення тест вводиться в структуру бази даних як новий. Після виконання новий тест переходить в категорію тестів, придатних для повторного використання або

застарілих. Якщо виконання тесту сприяло збільшенню поточної ступеня покриття коду, тест позначається як придатний для повторного використання. В іншому випадку він позначається як застарілий і відкидається. Існуючі тести, повторно запущені після внесення зміни в код, також класифікуються заново як придатні для повторного використання або застарілі залежно від тестових траєкторій і використовуваного критерію тестування.

Класифікація тестів по відношенню до змін в коді вимагає аналізу наслідків змін. Тести, що активують код, порушене змінами, можуть вимагати повторного запуску або виявитися застарілими-шими. Щоб тест був включений в клас тестів, які потребують повторного запуску, він має бути порушений змінами в коді, а також повинен сприяти збільшенню ступеня покриття зміненого коду по використовуваному критерію. Порушених елементом тесту може бути траєкторія, вихідні значення, або і те, і інше. Щоб тест був включений в клас тестів, придатних для повторного використання, він повинен робити внесок в збільшення ступеня покриття коду і не вимагати повторного запуску.

Ступінь покриття коду визначається для тестів, придатних для повторного використання, оскільки до цього класу належать тести, які не потребують повторного запуску і сприяють збільшенню ступеня покриття до бажаної величини. Якщо є компонент програми, що не задіяний придатними для повторного використання тестами, то замість них вибираються і виконуються з метою збільшення ступеня покриття тести, що вимагають повторного запуску. Після запуску такої тест стає придатним для повторного використання або застарілим. Якщо тестів, які потребують повторного запуску, більше не залишилося, а необхідна ступінь покриття коду ще не досягнута, породжуються додаткові тести і тестування повторюється.

Остаточний набір тестів збирається з тестів, придатних для повторного використання, тестів, що вимагають повторного запуску, і нових тестів. Нарешті, застарілі і надлишкові тести видаляються з набору тестів, оскільки надлишкові тести не перевіряють нові функціональні можливості і не збільшують покриття.

12.6. Можливості повторного використання тестів

До зміни існуючих тестів можуть привести три види діяльності програмістів:

1. Створення нових тестів.
2. Виконання тестів.
3. Зміна коду.

Оскільки кожен тест містить вхідні дані, вихідні дані і траєкторію, ці компоненти можуть піддатися зміні в будь-якій комбінації. При зміні вхідних даних існуючого тесту будемо вважати, що старий тест припиняє існування, і створюється новий тест. Таким чином, до числа дозволених змін тесту відносяться всі можливі пертурбації вихідних даних або траєкторій. Зміна вихідних даних без зміни траєкторії і / або вхідних даних Нево-Можливість запозичити. Отже, існує тільки два можливих варіанти зміни тесту: зміна траєкторії або зміна траєкторії і вихідних даних.

Згідно з наведеними вище міркуваннями можна виділити чотири рівні повторного використання тесту:

• *Рівень 1*: Тест не допускає повторного використання. Потрібне створення нового набору тестів (наприклад, шляхом видалення або зміни цього тесту).

• *Рівень 2*: Повторно використовувати можливо тільки вхідні дані тесту. У багатьох випадках мета тестування полягає в активізації деяких покриваються елементів програми. Якщо з траєкторії існуючого тесту видно, що елементи програми, які підлягають покриттю, задіюються до змінених команд, вхідні дані тесту можуть бути використані повторно для покриття цих елементів. В результаті змін в програмі і / або технічному завданні но-вая траєкторія і вихідні дані тесту можуть відрізнятися від результатів попереднього виконання. Таким чином, тести першого рів-ня повинні бути запущені повторно для отримання нових вихідних даних і траєкторій.

• *Рівень 3*: Можливо повторне використання як вхідних, так і вихідних даних тесту. Очевидно, що на цьому рівні зазвичай рас-покладаються функціональні тести. Якщо модуль піддався тільки зміни коду із збереженням функціональності, можливе повторне використання існуючих функціональних тестів для перевірки правильності реалізації. Оскільки траєкторія може вимірюватися, а вихідні дані - зазнати впливу з боку змін коду, такі тести повинні бути запущені повторно, але очікується отримання ідентичних результатів.

• *Рівень 4*: Найвищий рівень повторного використання тесту, який передбачає повторне використання вхідних даних, вихідних даних і траєкторії тесту. У цьому випадку на траєкторії тесту не змінюється ні один оператор. Отже, в повторному запуску цих тестів необхідності немає, так як вихідні дані і траєкторія залишаться незмінними.

12.7. Класифікація вибіркових методів

Для перевірки коректності різних підходів до регрессионному тестування використовується модель оцінки методів регресійного тестування. Основними об'єктами розгляду стали повнота, точність, ефективність і універсальність.

Точність - міра здатності методу уникати вибору тестів з T, на яких результат виконання зміненої програми не буде відрізнятися від результату її первісної версії, тобто тестів, які можуть виявляти помилки в P '.

*Припустимо, що набір T містить r регресійних тестів. З них для n тестів (n ≤ r) поведінку і результати виконання старої програми P відрізняються від поведінки і результатів виконання нової програми P '. Набір тестів T 'T містить m (m ≠ 0) тестів, отриманий-них з використанням методу відбору регресійних тестів M. З цих m тестів для l тестів поведінку P' і P різниться. Точність T 'відноси-кові P, P', T і M, виражена у відсотках, визначається виразом 100 * (1 / m), тоді як відповідний відсоток обраних тестів визначається виразом 100 * (l / n), якщо n ≠ 0 або дорівнює 100%, якщо n = 0.*

Виходячи з наведеного визначення, точність безлічі тестів - це відношення числа тестів даної множини, на яких результати виконання нової і старої програм розрізняються, до загальної кількості тестів безлічі. Точність є важливим атрибутом методу регресійного тестування. Неточний метод має тенденцію відбирати тести, які не повинні були бути обрані. Чим менш точний метод, тим ближче обсяг обраного набору тестів до обсягу вихідного набору тестів.

Ефективність - оцінка обчислювальної вартості стратегії вибіркового регресійного тестування, тобто вартості реалізації її вимог по часу і пам'яті, а також можливості автоматизації. Відносної ефективністю називається ефективність методу тестування за умови наявності не більше однієї помилки в тестованої програмі. Абсолютною ефективністю називається ефективність метода в реальних умовах, коли оцінка кількості помилок в програмі не обмежена.

Універсальність відображає міру здатності методу до застосування в досить широкому діапазоні ситуацій, що зустрічаються на практиці.

Для програми P, її зміненої версії P 'і набору тестів T для P потрібно, щоб методика вибіркового повторного тестування удов-летворяла наступними критеріями оцінки:

• Критерій 1 - *Безпека.*

Методика вибіркового повторного тестування повинна бути без-запасних, тобто повинна вибирати всі тести з T, які потенціалом-но можуть виявляти помилки (всі тести, чия поведінка на P 'і P може бути різним). Безпечна методика повинна розглядати наслідки додавання, видалення і зміни коду. При додаванні нового коду в P, в T можуть вже міститися тести, що покривають цей новий код. Такі тести необхідно виявляти і враховувати при відборі.

• Критерій 2 - *Точність.*

Стратегія повторного прогону всіх тестів є безпечною, але неточною. На додаток до вибору всіх тестів, потенційно здатних виявляти помилки, вона також вибирає тести, які ні в якому разі не можуть демонструвати змінений поведінка. В ідеалі, методика вибіркового повторного тестування повинна бути точною, тобто повинна вибирати тільки тести зі зміненим поведінкою. Однак для довільно взятого тесту, не запускаючи його, неможливо визначити, чи зміниться його поведінку. Отже, в кращому випадку ми можемо розраховувати лише на деяке збільшення точності. Всілякі існуючі вибіркові методи регресійного тестування розрізняються не в останню чергу вибором об'єкта або об'єктів, для яких виконується аналіз покриття та аналіз змін. Наприклад, при аналізі на рівні функції при зміні будь-якого оператора функції вся функція вважається зміненою; при аналізі на рівні окремих операторів ми можемо виключити частину тестів, що містять виклик функції, але не активують змінений оператор. Вибір об'єктів для аналізу покриття відбивається на рівні подробиці аналізу, а значить, і на його точності і ефективності. Абсолютні величини точності і кількості обраних тестів для заданих набору тестів і безлічі змін повинні розглядатися тільки разом зі зменшенням розміру набору тестів. Невеликий відсоток обраних тестів може бути прийнятним, тільки якщо рівень точності залишається досить високим.

• Критерій 3 - *Ефективність.*

Методика вибіркового повторно¬го тестування повинна бути ефективною, тобто повинна допускати автоматизацію і виконуватися досить швидко для практичного застосування в умовах обмеженого часу регресійного тестування. Методика повинна також передбачати зберігання інформації про хід виконання тестів в мінімально можливому обсязі.

• Критерій 4 - *Універсальність.*

Методика вибіркового повторного тестування повинна бути універсальною, тобто застосовується до всіх мов і мовних конструкцій, ефективної для реальних програм і здатною до обробки як завгодно складних змін коду.

У загальному випадку існує певний компроміс між безпекою, точністю і ефективністю. При відборі тестів аналіз необхідно провести за час, менше, ніж потрібно для виконання і перевірки результатів тестів з T, що не увійшли до T '. З урахуванням цього огра-ніченний рішенням завдання регресійного тестування буде безпечний метод з хорошим балансом дешевизни і високої точності.

Контрольні запитання

1. Визначте цілі і завдання регресійного тестування.
2. Проаналізуйте види регресійного тестування.
3. Проведіть обгрунтування коректності методу відбору тестів.
4. Проведіть класифікацію тестів при відборі.
5. Проаналізуйте можливості повторного використання тестів.
6. Проведіть класифікацію вибіркових методів.