

Процик Петро Павлович,
Аспірант, Київський Національний Університет ім. Т.Г. Шевченка,
факультет кібернетики, кафедра Теорії та технології програмування

Оцінка якості модульних тестів на основі метрик

Популярним сучасним методом розробки якісного прикладного програмного забезпечення є метод «розробка через тестування» (Test-Driven Development) [1]. Його особливість полягає у тому, що застосовується процес зворотній традиційному способу розробки, тобто, тест передує програмі, яку він тестує.

Таким чином, у формі виконуваних модульних [2] тестів фіксуються вимоги, припущення, обмеження та варіанти очікуваного використання майбутньої програми. Далі виконується реалізація цих вимог – створення коду програми, який буде задовольняти поточну множину тестів. Наступний крок починається з додавання нових тестів, і процес повторюється.

І хоча, при цьому, система перебуває на постійному контролі, проблема адекватності тестів залишається відкритою: наскільки фактична функціональність системи покрита тестами? Чи достатньо створених тестів? Чи є вони коректними?

Для відповіді на ці питання, спочатку формулюються критерії адекватності тестів. Традиційно вони зводяться до наступних властивостей, яким повинні задовольняти тести [3]:

Кожен тест повинен бути незалежним від інших тестів та інших тестових одиниць програми (Тестовою одиницею програми називають найменшу частину програми, яка підлягає модульному тестуванню). Властивість елементарності – окремий тест повинен призводити до проходження одного функціонального шляху на графі програми; повторюваність – модульні тести повинні виконуватись автоматично, не залежати від оточення та при повторному застосуванні призводити до одного результату; прагматичність – тести повинні забезпечувати належну роботу системи, що задокументована у вимогах.

За умови прийняття цих властивостей як базових критеріїв, у роботі запропоновано підхід для оцінки:

- 1) Поточної якості програмної системи;
- 2) Адекватності створених тестів;
- 3) Прогноз змін якості при додаванні нової функціональності.

При цьому, інтуїтивно зрозумілим є те, що чим більш складною є програма, тим більша кількість тестів необхідна для перевірки її коректності. Крім того, необхідно враховувати той факт, що у складних системах може бути багато шляхів, якими вона може потрапити з одного фіксованого стану в інший фіксований стан. Звідси випливає поняття функціонально різних тестів. Тести називають функціонально різними, якщо їх виконання призводить до проходження різних функціональних шляхів у графі програми.

Для оцінки складності програм використовується метрика цикломатичної складності. Вона досить часто використовується при тестуванні [4]. Метрика [5] – це емпірична, описувальна числова оцінка атрибутів об'єкту отримана на основі правил деякої моделі або формальної теорії. В контексті програмної інженерії, метрика програмного забезпечення – це міра, що дозволяє отримати числове значення деякої властивості програмного продукту або його специфікації.

Значення цикломатичної складності $C(G)$ керуючого графу G підраховується за такою формулою: $C(G) = E - N + 2$, E – кількість ребер, N – кількість вершин і дорівнює кількості лінійно незалежних шляхів у програмі. Значимість цієї метрики для оцінки якості програми підтверджується рядом проведених досліджень. Так, наприклад, у 2008 році компанією Enerju [6] було виконано кореляційний аналіз Java програм з відкритими кодами (за даними джерела десятки тисяч файлів) на основі кількості виправлених помилок у цих програмах та значеннях цикломатичної складності. На основі цього аналізу були розроблені рекомендації, щодо допустимих значень цикломатичної складності на один тестовий модуль: $1 < C(G) < 10$ – малий ризик помилки, $11 < C(G) < 20$ – середній, $21 < C(G) < 50$ – високий, $51 < C(G)$ – такі модулі майже гарантовано містять суттєві функціональні недоліки.

На базі цих даних як **результати та висновки роботи**, пропонується наступне:

- 1) Критерій адекватності тестів: відношення загальної кількості функціонально різних тестів U до сумарної цикломатичної складності програми C повинне дорівнювати фактичному покриттю коду програми тестами P : $A = 100\% \times U / C \approx P$.
- 2) Далі на основі статистичних даних, можна наближено обчислити кількість не знайдених функціональних помилок у програмі: $D = (1 - P) \times C / 50$. При цьому фактичне значення скоріш за все буде більшим, тому що наявність тесту не гарантує відсутності помилок.

- 3) При додаванні нової функціональності до програми для того, щоб якість системи залишалась на тому ж рівні, необхідно, щоб відношення кількості нових тестів до складності нового коду прямувала до одиниці: $U_1 / C_1 \rightarrow 1$.

На основі даного підходу було проведено вивчення декількох систем з відкритими кодами (з сайту <http://www.codeplex.com>) та були отримані наступні результати:

Назва	Строк коду	Цикломатична Складність	Покриття коду	Кількість тестів	A	D
MEF	4559	2203	94.32	1510	69%	3
FishDawg	1850	647	98.04	809	125%	0
AddIn Application Framework	3318	2023	70.73	654	32%	12

Можна зробити висновок, що перша та друга системи, повинні мати досить високий рівень якості. Для другої системи приблизно 25% усіх тестів дублюють функціональність інших та не є необхідними. Тести третьої системи не достатньо відповідають визначеним критеріям і потребують модифікації.

ЛІТЕРАТУРА

1. Kent Beck, Test Driven Development: By Example, Addison-Wesley Professional, 2002, – 240 p.
2. Брауде Э. Дж. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655 с.:ил.
3. Andrew Hunt, Pragmatic Unit Testing in C# with NUnit, The Pragmatic Programmers, 2004. – 176 p.
4. Thomas J McCabe, A complexity measure, IEEE Transactions on Software Engineering, VOL.SE-2, NO.4, 1976.
5. Cem Kaner, Walter P. Bond, Software Engineering Metrics: What do they measure and how do we know?, 10TH International Software Metrics Symposium, Metrics 2004.
6. Enerjy, McCabe Cyclomatic Complexity: the proof in the pudding, <http://www.enerjy.com/blog/?p=198>