ДОПОВІДІ
НАЦІОНАЛЬНОЇ
АКАДЕМІЇ НАУК
УКРАЇНИ

6 • 2007

УДК 681.1

М. В. Дідковська

Y-подібна модель життєвого циклу програмного забезпечення

(Представлено академіком НАН України В. Н. Редьком)

The article outlines a Y-shaped model of software life cycle that considers the testing on each stage of the software development. A software test classification based on the proposed Y-shaped model is presented.

Програмне забезпечення (ПЗ) в процесі своєї розробки й експлуатації проходить ряд певних етапів: виникнення та дослідження ідеї, аналіз вимог і проектування, безпосередньо кодування, тестування та налагодження, введення програми в дію, експлуатація та супровід, виведення з експлуатації [1]. Залежно від обраної моделі життєвого циклу (ЖЦ) ПЗ, ці фази можуть бути розбиті на декілька складових частин або об'єднані. Модель ЖЦ ПЗ схематично пояснює, яким чином будуть виконуватися дії з розроблення програмного продукту за допомогою опису "послідовності" цих дій. Така послідовність може бути як лінійною, так і нелінійною, оскільки фази можуть слідувати одна за іншою, повторюватися або відбуватися одночасно. Найбільш відомими та широко використовуваними моделями життєвого циклу ПЗ є: каскадна, V-подібна, еволюційна, прискорена, прототипування, швидка розробка, інкрементна та спіральна моделі.

На сьогоднішній день велика увага приділяється тестуванню ПЗ. Сучасні технології проектування вимагають, щоб процес тестування починався на ранніх фазах ЖЦ ПЗ. Найбільшу увагу процесам тестування приділено в V-подібній моделі ЖЦ ПЗ.

Розглянемо її докладніше.

V-подібна модель життєвого циклу розробки ПЗ. V-подібна модель була створена для допомоги працюючій над проектом команді в плануванні та забезпечення подальшої можливості тестування системи. В цій моделі особливе значення надається діям, спрямованим на верифікацію й атестацію продукту. План іспиту приймання кінцевого програмного продукту замовником розробляється на етапі планування, а компоновочного іспиту системи — на фазах аналізу, розробки проекту тощо. Процес розробки планів тестування позначений пунктирною лінією між прямокутниками V-подібної моделі (рис. 1).

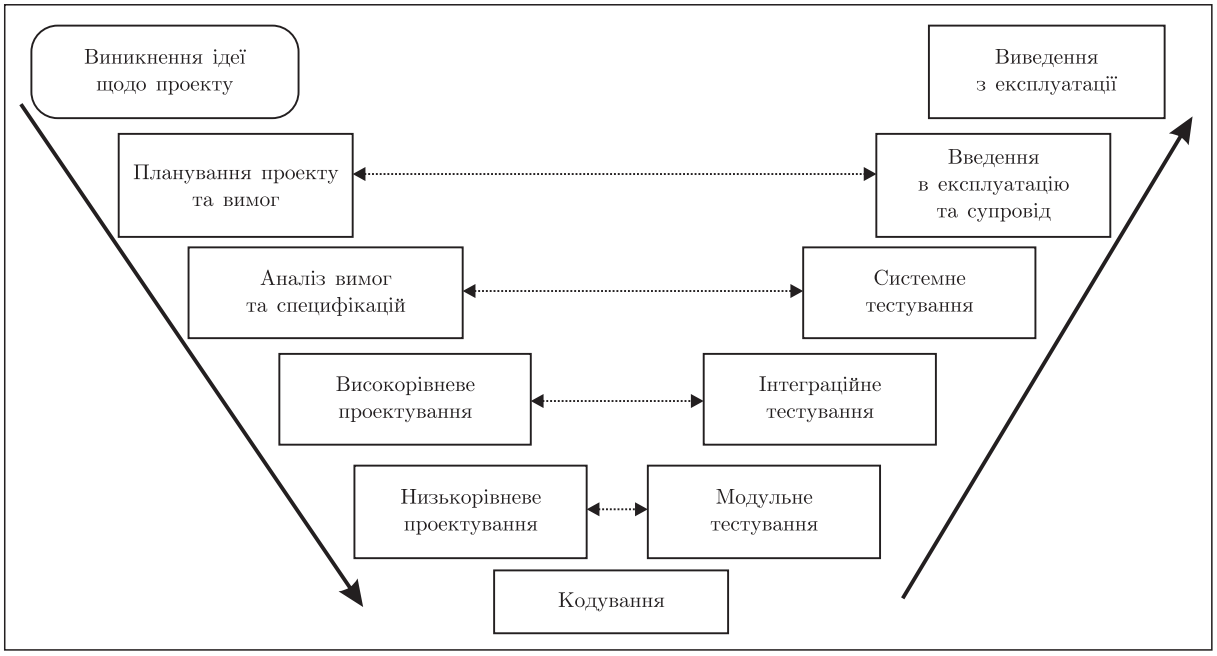

Рис. 1. V-подібна модель життєвого циклу програмного забезпечення

Проте, незважаючи на всі свої переваги та спрямованість на тестування, V-подібна модель має послідовну структуру: кожна наступна фаза починається лише по завершенню попередньої. Незважаючи на те що передбачається план тестування виробляти на ранніх етапах ЖЦ ПЗ, саме тестування здійснюється вже після того, як ПЗ створено. Отже, помилки, виявлені на цьому етапі та пов'язані з неправильним проектуванням, спричинять повернення на попередні етапи ЖЦ та значні фінансові втрати. При використанні UML-базованих критеріїв інтеграційного тестування, наведених у [2], з'явилася можливість проведення тестів до того, як створений код ПЗ — ще на етапах аналізу вимог і високорівневого проектування. Тому пропонується така модифікація моделі ЖЦ ПЗ.

Y-подібна модель життєвого циклу розробки ПЗ. Спочатку виникає ідея про створення проекту — дана фаза неформалізована, що відображується на схемі (рис. 2) пунктирним прямокутником з округленими кутами.

На етапі формалізації специфікацій з'являється можливість спроектувати тести на відповідність системи своїм функціональним вимогам та провести первинну атестацію ПЗ, тобто переконатися в тому, що проект відповідає поставленим замовником цілям.

Таким чином, паралельно здійснюється фаза планування й аналізу вимог і специфікацій і фаза тестування вимог і специфікацій, що відображено на схемі (див. рис. 2) двосторонньою стрілкою.

На підставі наявних специфікацій здійснюється проектування ПЗ. За допомогою критеріїв інтеграційного тестування [2] здійснюється тестування самого проекту, його внутрішньої логіки. Тому паралельно з фазою проектування здійснюється фаза інтеграційного тестування. При проведенні кодування ПО здійснюється його паралельне тестування — всіх реалізованих методів, класів, інтеграції між ними і вже усього ПЗ в цілому в середовищі. Нарешті ПЗ вводиться в експлуатацію, супроводжується і, нарешті, по завершенню своєї актуальності, виводиться з експлуатації. Схема послідовності етапів, які ПЗ проходить на

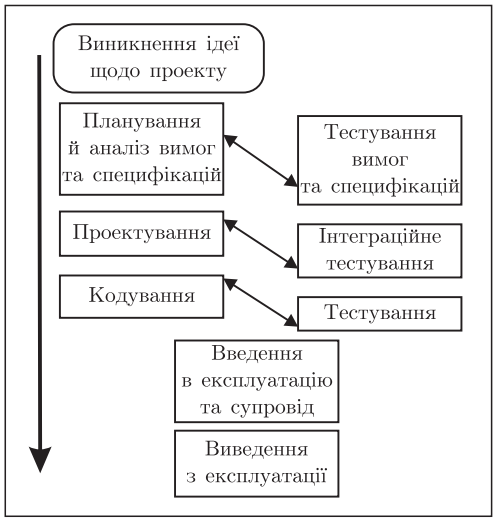

Рис. 2. Y-подібна модель життєвого циклу програмного забезпечення

шляху свого розвитку, наведена на рис. 2 і має форму латинської літери Y, тому запропонована модель ЖЦ ПЗ й одержала назву Y-подібної. Модель орієнтована на проведення тестування на кожному етапі ЖЦ ПЗ, тому пропонується зробити таку класифікацію тестів.

Класифікація тестів на основі ЖЦ ПЗ. Пропонується класифікувати множину тестів, беручи за основу етапи ЖЦ ПЗ та спускаючись по них зверху вниз, при цьому деталізуючи тести попередніх рівнів на іншому рівні абстракції.

1. На етапі планування й аналізу вимог і специфікацій створюються UML діаграми прецедентів. Відповідно, першою групою тестів є тести, спроектовані для кожного прецеденту, для кожної взаємодії прецеденту з кожної з груп користувачів і для кожного зв'язку і послідовностей зв'язків на діаграмі прецедентів.

2. На етапі проектування ПЗ вже представлено у формальному вигляді за допомогою діаграм UML класів, станів, взаємодії тощо. Відповідно, до цієї групи тестів відносяться всі тести, які спроектовані на основі цих діаграм, з урахуванням критеріїв інтеграційного тестування, запропонованих у [2]. Дані тести в свою чергу можуть бути розбиті на підгрупи, відповідно до ієрархічної системи критеріїв, а саме: тести для об'єктів, тести для послідовної взаємодії об'єктів, тести для паралельної взаємодії об'єктів. Під об'єктами в даному випадку розуміються такі елементи UML діаграм, як переходи, активізації й операції інтерфейсу.

3. На етапі кодування має сенс виділяти групу тестів, пов'язаних безпосередньо з реалізацією коду і внутрішньою логікою програми. Дану групу можна розбити на дві підгрупи — проведення функціонального та структурного тестування. Функціональне тестування проводиться на основі методу аналізу граничних умов й еквівалентного розбиття. Тести для структурного тестування можуть бути розбиті на тести для потоку керування й тести для потоку даних. У свою чергу, надалі можуть бути виділені тести для покриття операторів, покриття умов, покриття циклів, шляхів тощо, відповідно до ієрархічної системи критеріїв структурного тестування. Крім того, на даному етапі можна використати класифікацію, запропоновану Майерсом [3]: тести для звертання до даних, обчислень, опису даних, порівняння, передачі керування, введення — виведення, інтерфейсу та ін.

4. На етапі введення програми в дію і її експлуатації пропонується виділити групу тестів на базі методу випадкового тестування.

Таким чином, в роботі наведено Y-подібну модель ЖЦ ПЗ, яка відображує паралельне проведення процесів формалізації вимог до проекту, проектування і кодування та відповідних їм фаз тестування. На основі цієї моделі запропонована класифікація тестів.

1. *ISO* 12207: 1995. – (ГОСТ Р – 1999). ИТ. Процессы жизненного цикла программных средств.
2. *Didkovska M.* Criteria for integration testing of component-based software // Электроника и связь. – 2004. – No 23. – С. 90–94.
3. *Майерс Г.* Искусство тестирования программ / Пер с англ. под ред. Б. А. Позина. – Москва: Финансы и статистика, 1982. – 172 с.

НТУ України "Київський політехнічний інститут"

Надійшло до редакції 25.10.2006

УДК 519.6

А. Г. Каграманян, В. П. Машталир, Е. В. Скляр, В. В. Шляхов

Метрические свойства разбиений множеств произвольной природы

(Представлено членом-корреспондентом НАН Украины Ю. Г. Стояном)

The interpretation of data content is closely connected with partition analysis. Different applications require different detailings of data partitions. For a system to be successful in a variety of problems, several partitions have to be ensured for cognitive-like techniques. A rational combination of low-level and high-level capabilities seems to be the most promising way to significantly improve the data understanding integrally. To reduce the gap between low-level features and high-level semantics in clustering, we propose, ground, and explore a new metric on partitions of an arbitrary measurable set.

В задачах распознавания образов факторизация информации в том или ином признаковом пространстве концептуально является одним из основных методов, лежащих в основе интерпретации данных. С одной стороны, может требоваться идентификация объектов, процессов или явлений с точностью до заданного или найденного в процессе анализа отношения эквивалентности. С другой, построение классов эквивалентности — часто суть и цель обработки данных для последующего этапа тематической трактовки отдельных регистрируемых представителей или даже фактор-множеств. В качестве типичного примера можно указать традиционную кластеризацию данных [1], особенно с целью дальнейшего компаративного распознавания [2]. Более детальным примером может служить сегментация изображений, т. е. построение разбиения поля зрения, удовлетворяющее условию принадлежности носителя "области интереса" одному классу эквивалентности [3]. В силу существенной неопределенности входных данных и возможности применения различных алгоритмов можно получать различные результаты, в частности, чрезмерную (объект расположен в нескольких факторах) или недостаточную (в классе эквивалентности находятся изображения фона) сегментацию.