УДК 004.056

Д.А. СЕМЧЕНКО, А.А.ЗАМУЛА, канд. техн. наук

МЕТОДЫ АНАЛИЗА ПРОЦЕССОВ КОНТРОЛЯ КАЧЕСТВА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ, ИСПОЛЬЗУЕМЫЕ ПРИ ПРОЕКТИРОВАНИИ И РАЗРАБОТКЕ СИСТЕМ ОБРАБОТКИ КРИПТОГРАФИЧЕСКОЙ ИНФОРМАЦИИ

Технические термины в области программного обеспечения (ПО) определены в стандартах ISO/IEC 9126-1, IEEE/IEC 12207, IEEE 1028 [1 – 3], что позволяет однозначно сформулировать используемую терминологию. В стандарте ISO/IEC 9126-1[4] определены соответствующие характеристики качества, при которых фундаментальный процесс тестирования состоит: из планирования и контроля, анализа и проектирования, внедрения и сопровождения оценки критериев для завершения тестирования и создания отчетной документации. Процесс тестирования базируется на сопоставлении ожидаемого (описанного в технической документации) и фактического (полученного в ходе испытаний) результата. В случае, когда результаты процессов совпадают, тест считается пройденным успешно.

Для достижения структурированной и управляемой разработки программного обеспечения используются модели разработки ПО. На данный момент известны и активно используются фундаментальные модели (модель «водопада» и V-модель) [5] и пошаговые модели (RAD, RUP, XP, SCRUM, DSDM) [6]. Важную роль в процессе разработки и тестирования ПО играет психологический аспект. Человек не склонен замечать свои собственные ошибки. Этот факт относится к аспектам организации и проведения процесса тестирования[7]. Поэтому процессы разработки и тестирования должны быть организованы независимо друг от друга.

Для получения объективной оценки о качестве ПО, с точки зрения функциональности, надежности, удобства использования, эффективности и гибкости к модификациям, необходимо использовать различные методы, которые можно разделить на две большие группы: статического и динамического анализа.

Под статическим анализом понимают анализ программного обеспечения, производимый без реального выполнения исследуемых программ. Как правило, важность методов статического анализа недооценивается, однако полученные результаты позволяют оптимизировать процесс разработки и обнаружить ошибки и отклонения от технической документации, стандартов и плана проекта. Кроме того, статический анализ может быть использован в целях выявления проблем безопасности (например, отсутствие защиты от переполнения буфера или отсутствие проверки входных данных на допустимые значения).

Ключевыми понятиями статического анализа являются экспертная оценка и инспекция (обзор). Существует четыре основных типа обзора: анализ со стороны руководства, технический анализ, информационный анализ, пошаговый анализ [8].

Количественные характеристики качества при статическом анализе могут быть получены путем использования метрик [9]. Метрики сложности программ принято разделять на три основные группы.

Метрики первой группы базируются на определении количественных характеристик, связанных с размером программы, и отличаются относительной простотой. К наиболее известным метрикам данной группы относится набор метрик Холстеда [9].

Основу метрики Холстеда составляют четыре измеряемые характеристики программы:

- NU_{Oprtr} (Number of Unique Operators) – число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций;
- NU_{Oprnd} (Number of Unique Operands) – число уникальных операндов программы;
- N_{Oprtr} (Number of Operators) – общее число операторов в программе;
- N_{Oprnd} (Number of Operands) – общее число операндов в программе.

На основании этих характеристик рассчитываются оценки длины программы, объема программы, сложности программы:

- длина программы (Halstead Program Length, HP_{Len}) $HP_{Len} = N_{Oprtr} + N_{Oprnd}$;
- объем программы (Halstead Program Volume, HP_{Vol}): $HP_{Vol} = HP_{Len} \log_2 HP_{Voc}$;
- сложность программы (Halstead Difficulty, H_{Diff}): $H_{Diff} = (NU_{Oprtr}/2) \times (N_{Oprnd} / NU_{Oprnd})$.

Метрики второй группы ориентированы на анализ исходного кода программы. Представителем данной группы является метрика цикломатической сложности по Мак-Кейбу, которая базируются на основе графа управляющей логики, в котором вычислительные операторы или выражения представляются в виде узлов, а передача управления между узлами – в виде дуг (см. рис. 1)

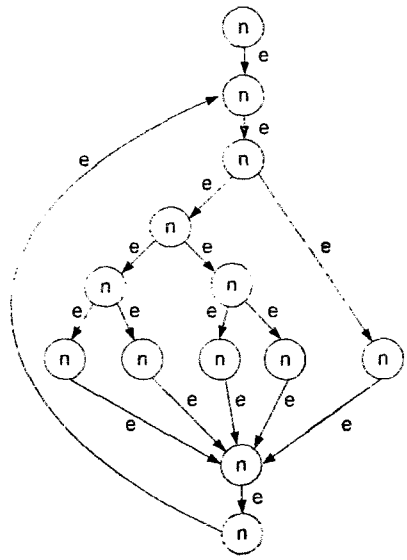

Рис. 1. Вид графа управляющей логики

Показатель цикломатической сложности позволяет оценить трудоемкость реализации отдельных элементов программы и скорректировать общие показатели оценки длительности и стоимости проекта. Упрощенная формула вычисления цикломатической сложности представляется следующим образом:

$$C = e - n + 2,$$

где e – число ребер, а n – число узлов на графе управляющей логики.

Показатель цикломатической сложности по Мак-Кейбу показывает требуемое количество проходов для покрытия всех связей в графе или количество тестовых прогонов программы, необходимых для наиболее полного тестирования.

Метрики третьей группы базируются на оценке использования, конфигурации и размещения данных в программе. К данной группе относятся метрика Чепина [9].

Суть метрики состоит в оценке информационной стойкости отдельно взятого программного модуля с помощью анализа характера использования переменных из списка входных и выходных переменных (список ввода-вывода). Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

- множество «P» – вводимые переменные для расчетов и для обеспечения вывода;
- множество «M» – модифицируемые или создаваемые внутри программы переменные;
- множество «C» – переменные, участвующие в управлении работой программного модуля;
- множество «T» – не используемые в программе переменные.

Значение метрики Чепина рассчитывается по формуле

$$Q = a_1 P + a_2 M + a_3 C + a_4 T,$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты, которые используются для отражения различного влияния на сложность программы каждой функциональной группы.

Использование статических методов анализа дает возможность уменьшить затраты на проведение тестирования за счет использования метрик. Результаты использования метрик должны быть получены до начала обзора и должны служить входными данными для экспертной оценки.

Динамический анализ представляет собой процесс оценки поведения системы или компонента системы (тест-объекта) во время выполнения (например, производительности памяти, использования процессора).

Существует несколько подходов для проведения тестирования тест-объекта, которые можно разделить на две группы: тестирование черного ящика и тестирование белого ящика.

При тестировании черного ящика, компонент системы рассматривается как объект внутреннее устройство которого неизвестно [10]. Тестовые данные берутся из спецификации тестового объекта. Эффективно применять такие техники тестирования черного ящика: анализ эквивалентных классов, анализ граничных значений, построение причинно-следственных связей, построение диаграмм прецедентов.

Основу техники анализа эквивалентных классов составляют два положения:

- исходные данные необходимо разбить на конечное число классов эквивалентности. Классом эквивалентности C(a) элемента a называется подмножество элементов, эквивалентных a. Из приведенного определения следует, что, если $b \in C(a)$, то C(a) = C(b)). В одном классе эквивалентности содержатся такие тесты, что, если один тест из класса эквивалентности обнаруживает некоторую ошибку, то и любой другой тест из этого класса эквивалентности должен обнаруживать эту же ошибку;

- каждый тест должен включать, по возможности, максимальное количество классов эквивалентности, чтобы минимизировать общее число тестов.

Разработка тестов данной техникой осуществляется в два этапа: выделение классов эквивалентности и построение теста. Классы эквивалентности выделяются путём выбора каждого входного условия, которые берутся из технического задания или спецификации.

Техника анализа граничных значений представляет собой определение максимальных и минимальных значений на границах классов эквивалентности. Разработка тестов этой техникой осуществляется путем составления позитивных (используются значения, принадлежащие классу эквивалентности) и негативных тестов (используются значения, не принадлежащие рассматриваемому классу эквивалентности).

Данная техника отображает взаимосвязь между причинами возникновения ошибки и их влиянием на компоненты системы (см. рис. 2).

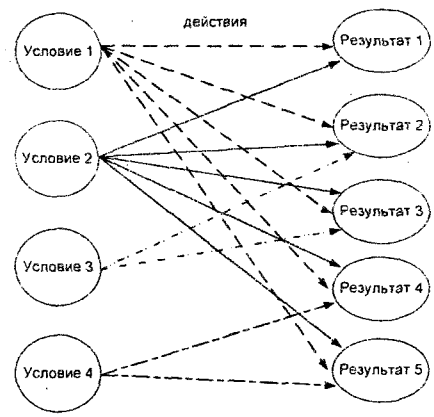

Рис. 2. Отображение причинно-следственных связей на графе

Разработка тестов этой техникой осуществляется путем разбиения спецификации на части, для каждой из которых определяется множество причин и следствий. Под причиной понимается отдельное входное условие или класс эквивалентности. Следствие представляет собой выходное условие или преобразование системы. На основе анализа семантического (смыслового) содержания спецификации строится граф, в котором последовательно перебираются всевозможные комбинации причин и следствия для них. После построения графа делаются выводы о том, какие условия должны быть объединены в отдельные тесты.

Техника построения диаграммы прецедентов используется для отображения внешнего вида системы с точки зрения пользователя и для моделирования взаимосвязей между различными подсистемами.

Основой для тестирования белого ящика служит исходный код тестового объекта. Поэтому, техники тестирования белого ящика также называют структурным тестированием. Основная идея тестирования заключается в выполнении каждой части кода тестового объекта хотя бы один раз. Операторы и условия которые используются в тестируемой части кода представляются в виде блок-схем (см. рис. 3).

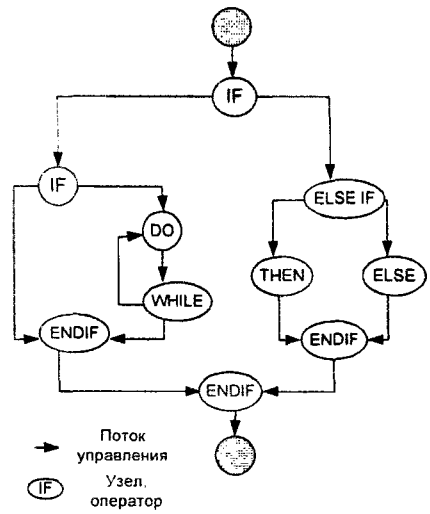

Рис. 3. Фрагмент кода программы

Использование блок-схем упрощает задачу создания тестов за счет наглядного представления механизма функционирования программного кода, позволяя наиболее полно найти существующие недостатки за минимальное время.

Для достижения высоких показателей качества ПО необходимо исчерпывающее тестирование, то есть тестирование над всеми возможными наборами данных. В большинстве случаев это невозможно из-за недостатка времени на проведение тестирования. Решением данной проблемы служит совокупное использование рассмотренных методов, выбор наиболее приоритетных тестов (по результатам каждого метода составляется набор тестов) в соответствии с функциональными требованиями, выдвигаемыми к системе.

Методы и техники тестирования необходимо выбирать в зависимости от сложности структуры системы, требований, которые выдвинуты к системе, и временных рамок, в пределах которых система должна быть введена в эксплуатацию.

При планировании и проведении тестирования необходимо учитывать требования: соответствие стандартам, результаты проведенного анализа рисков системы, опыт персонала; наличие инструментария для проведения автоматического тестирования и технической документации по ней.

При управлении процессом тестирования необходимо осуществлять планирование, мониторинг и контроль на различных циклах испытаний. Проведение тестирования осуществляется согласно документации – плану тестирования. В плане тестирования должна быть

описана стратегия (цели, меры, средства) тестирования и приоритетность проведения испытаний, основанная на анализе рисков. Анализ рисков представляет собой анализ вероятности появления ошибок после проведения испытания. Количество и интенсивность испытаний должны быть адаптированы с конкретными потребностями проекта.

Список литературы: 1. *ISO/IEC* 9126-1:2001, Software Engineering. – Product quality – Part 1: Quality model, Quality characteristics and sub-characteristics. 2. *IEEE/EIA* Std 12207-1996: Information Technology – Software life cycle processes. 3. *IEEE* Std 1028-1996: IEEE. 4. *ISO/IEC* 9126-1:2001, Software Engineering. – Product quality – Part 1: Quality model, Quality characteristics and sub-characteristics. 5 *Boehm B. W: "A Spiral* Model of Software Development and Enhancement", ACM SIGSOFT, August 1986, pp. 14-24. 6. *Spillner, A.* From V-model to W-model – Establishing the Whole Test Process / Proceedings. 7. *IEEE* Std 829-1998, IEEE Standard for Software Test Documentation (under revision, new edition probably in 2006). 8. *Beizer, B.* Software Testing Techniques, Van Nostrand Reinhold, 1990. 9. *Fenton, N. E.:* Software Metrics, Chapman&Hall, 1991. 10. *Beizer, B.* Black-Box Testing, John Wiley & Sons, 1995. 11. *Fenton, N. E.:* Software Metrics, Chapman&Hall, 1991. Conquest 2000 – Workshop on "Testing Non-Functional Software Requirements", Sept. 2000, Nuremberg, pp. 221-231.

Харьковский национальный
университет радиоэлектроники

Поступила в редколлегию 12.07.201.