Об усовершенствовании статистического метода оценки полноты тестов программ и устройств

Басок Б. М., Гречин А.А.[1]

[1]Московский государственный технический университет радиотехники, электроники и автоматики,
119454, Россия, г. Москва, Проспект Вернадского, 78
vm_e@mail.ru

Аннотация. Рассматривается статистический метод проверки полноты тестов, основывающийся на анализе результатов работы программ и математических моделей дискретных устройств с внесёнными дефектами. В работе предлагается существенно сократить временные затраты при оценке качества тестов данным методом, с одной стороны, путём внесения кратных дефектов, с другой, – путём взаимного использования результатов анализа тестов объекта тестирования и отдельных его компонентов. Приводятся экспериментальные данные, подтверждающие эффективность предлагаемых подходов. *Ключевые слова:* полнота тестов, анализ тестов, кратные отказы, мутации.

1 Введение. Постановка задачи

Важнейшей характеристикой тестов программных систем (ПС) и дискретных устройств (ДУ) является их полнота. Под полнотой теста или его проверяющей способностью принято понимать [1,2] выраженное в процентах отношение выявляемых данным тестом отказов заданного класса к общему их числу. От полноты тестов напрямую зависит качество контролируемого объекта. Поэтому задача синтеза тестов высокой полноты является одной из важнейших задач контроля любого продукта на всех этапах его разработки и эксплуатации. При этом следует отметить, что, как правило, тесты ПС и существенная часть тестов ДУ создаются и отлаживаются вручную, и разработка новых дополнительных тестов для повышения полноты контроля требует больших затрат времени и средств.

Обычно при проверке полноты функциональных тестов ПС используются методы, основывающиеся на покрытии кода программы или методы, базирующиеся на оценке количества охваченных тестом отдельных функций ПС [3].

Наряду с данными методами для проверки полноты тестов используется метод, основывающийся на анализе результатов выполнения тестов для ПС с искусственно введенными одиночными постоянными дефектами из списка отказов заданного класса и исходной ПС. При этом предполагается, что данный метод применяется после того, когда все собственные дефекты, обнаруженные данными тестами, исправлены, и влияние не обнаруженных ими собственных

дефектов ПС невелико. Указанный метод является некоторым аналогом классического метода, применяемого при анализе тестов ДУ и основывающегося на сравнении результатов моделирования исправной и неисправной ДУ[4].

Известны два подхода при реализации данного метода. Первый из них [5,6] основывается на внесении дефектов непосредственно в объектный код ПС. Такими дефектами могут быть, например, инверсия бита или байта объектного кода программы.

Второй подход, получивший название мутационного анализа [7-10], основывается на внесении в исходный текст программы незначительных дефектов (мутантов). По определению [7] *мутант* – это одиночное синтаксически корректное изменение подлежащей тестированию ПС. Специальные программы – генераторы мутантов - формируют их списки. Мутанты, входящие в список, осуществляют либо замещение констант, либо замещение переменных, либо замещение операторов. Генераторы мутантов строятся на основе синтаксического анализа исходного кода ПС[7].

Первый подход применим, в первую очередь, при анализе тестов ПС, для которых отсутствует доступ к исходному коду.В то же время, для обеспечения безопасности работы эксплуатируемых ПС необходимо убедиться в высокой степени их надежности. Поэтому в любой момент может возникнуть необходимость в оценке качества предоставленных пользователю функциональных тестов, поскольку в редких случаях технологии разработки ПС, применяемые разработчиками включают обязательный этап тестирования безопасности [6].

Второй метод особенно полезен, поскольку с его помощью можно точно установить место внесенного дефекта и разрабатывать дополнительные тесты для выявления дефектов, не обнаруженных анализируемыми тестами. Кроме того, при реализации второго подхода, в отличие от первого, отсутствует опасность того, что внесение искусственного дефекта в ПС исказит операционную среду. Впрочем, данный недостаток можно преодолеть путем запуска ПС в системе виртуальных машин. При этом отказ, внесение которого приводит к искажению операционной среды, можно считать выявляемым.

Описанные подходы применимы и при тестировании поведенческой модели ДУ на языке описания аппаратуры HDL, в частности на VHDLи Verilog,и анализе собственно тестов ДУ, представленных этой моделью [9,11].Например, в [9] описаны принципы разработки генератора мутаций для ПС на VHDL.

Главным недостатком, как первой, так и второй реализации рассматриваемого метода оценки полноты тестов ПС являются чрезвычайно большие затраты времени, как следствие наличия большого количества мутантов или возможных искажений разрядов объектного кода ПС. Даже при тестировании сравнительно небольших ПС их количество может достигать многих десятков тысяч. При тестировании программ, содержащих несколько тысяч операторов, это число может многократно увеличиться. Например, при тестировании программы, реализующей метод Монте-Карло, генератор выдал более девятисот тысяч мутантов [10].

В некоторых работах [7] были рассмотрены методы сокращения множества

рассматриваемых мутантов путем выделения групп эквивалентных мутантов и при оценке полноты тестов анализировать только одиночных представителей этих групп. Однако, как показано в [7], список анализируемых отказов сокращается не более чем на 8% - 10%.

Другой подход, основывается на замене общего списка мутантов некоторой его выборкой, как правило, это некоторая часть общего списка, например, в [7] случайным образом выбирается 10% общего списка дефектов. Тем не менее, выбранный список оказывается достаточно большим. Кроме того в подобных работах отсутствует обоснование размеров выборки. В этом случае обычно полагаются на накопленный в данной области опыт.

Для существенного сокращения времени при реализации метода оценки полноты в работе [12] предложен и обоснован способ, известный как статистический. В соответствии с этим способом при анализе тестов используется не полное множество отказов, а некоторая небольшая случайная выборка. Согласно [12] для случайной выборки из 400 отказов заданного класса можно гарантировать, что полнота тестового набора для всей совокупности отказов заданного класса отличается не более чем на 5% от истинного значения с достоверностью 0,95. Величина выборки не зависит от сложности и размеров объекта тестирования.

Таким образом, количество модифицированных неисправных копий анализируемого объекта (ПС или ДУ) оказывается фиксированным и сравнительно небольшим.

Статистический способ оценки полноты был программно реализован и показал свою эффективность при многолетней эксплуатации программного комплекса моделирования, синтеза и анализа тестов, разработанного в ИНЭУМ и внедренного в ряде организаций, а также при анализе полноты тестов СБИС повышенной сложности с помощью первого в Советском Союзе многопроцессорного программно-аппаратного комплекса-ускорителя логического моделирования, разработанного в ИНЭУМ [13].

В работах [5,11] обсуждаются возможности применения статистического способа оценки полноты при мутационном подходе и искажении объектного кода программы.

Однако, несмотря на ряд преимуществ статистического способа проверки полноты абсолютная величина временных затрат при его использовании остается весьма значительной. В первую очередь, это касается затрат времени при анализе тестов программ с невысокой проверяющей способностью. Так при анализе пяти тестов программы размер исполнительного модуля которой составляет 18 кбайт, понадобилось 1708 прогонов данной программы. Если предположить, что время выполнения программы составляет одну минуту, то временные затраты составят более 28 часов. При этом несмотря на то, что имеется такой источник сокращения времени, как возможность распараллеливания процесса оценки полноты, к примеру на нескольких компьютерах, величина временных затрат остается существенной.

В данной работе рассматриваются два подхода к сокращению времени оценки статистической полноты тестов. Первый подход основывается на анализе тестов в классе одиночных константных отказов путём внесения кратных дефектов из данного класса. Второй подход базируется на совместном

анализе интеграционных тестов объекта контроля и тестов отдельных его частей.

2 Метод внесения кратных отказов

В работе одного из авторов данной статьи [14] был предложен метод анализа тестов ДУ, называемый методом моделирования кратных отказов. В соответствии с этим методом процесс проверки полноты тестов можно ускорить, если вносить не одиночные, а кратные отказы. Последние можно получить путём случайного разбиения списка невыявленных предыдущими тестами отказов на небольшие непересекающиеся группы одного размера (возможно последняя из групп окажется меньше). Отказы объединяются в группы случайным образом. При этом в соответствии с гипотезой о редкой компенсации кратных отказов [1,11] считается, если тестом не выявляется полученный таким образом кратный отказ, то не выявится и не один из одиночных дефектов, входящих в данную группу. В том случае, если состояния выходов модели объекта тестирования с внесённым кратным отказом отличаются от состояний одноименных выходов модели исправного объекта, следует проверить возможность выявления тестом дефектов, образующих кратный отказ по одному.

В работе [14] обосновывается близкий к оптимальному размер групп для списка дефектов, оставшихся не выявленными после анализа группы первых тестов и относящихся к классу трудно проверяемых отказов. В соответствии с приводимыми в работе [14] оценками применение указанного метода повысит скорость анализа качества тестов в несколько раз.

В докладе применяется подход, использующий, с одной стороны, статистический способ проверки полноты тестов, с другой, – метод кратных отказов. Этот подход применим как для ПС, так и для ДУ. В соответствии с данным подходом не выявленные очередным тестом отказы из выборки [5,12] разбиваются на группы. Размер группы определяется исходя из минимизации математического ожидания случайной величины количества прогонов теста [14]. Общее количество прогонов теста определяется как сумма двух слагаемых: количества прогонов равных количеству полученных групп отказов и количества прогонов равного количеству одиночных дефектов, образующих обнаруженные кратные дефекты.

Таким образом, величина группы при анализе полноты теста *i* может быть определена путем минимизации функции, имеющей следующий вид:

$$F(G_i) = \frac{1}{G_i} + 1 - \prod_{j=1}^{Gi} \left(\frac{N_i - L_{i-1} - j + 1}{N_i - j + 1} \right) \qquad (1)$$

где L_{i-1} - количество выявленных дефектов на предыдущем (i-1)-ом тесте, N_i – количество дефектов заданного класса, оставшихся невыявленными после анализа предыдущих (i-1) тестов.

Размер группы для анализа качества первого теста можно принять равным единице, учитывая тот факт, что первые тесты обычно обладают хорошими

проверяющими свойствами. С другой стороны, как показали эксперименты, эту величину можно увеличить до трёх.

Если очередной тест не выявил ни одного отказа, то переменной в функции (1) можно присвоить не равное нулю значение аналогичной переменной, определенное ранее для ближайшего из предыдущих тестов.

Данный подход был апробирован при анализе тестов группы промышленных консольных приложений (консольный тип был выбран для удобства автоматизации процесса проверки полноты), работающих в операционной среде Windows. В качестве моделей отказов использовались постоянные дефекты типа инверсии бита/байта. Результаты анализа тестов методами одиночных и кратных отказов с использованием статистической выборки моделируемых отказов показали, что количество прогонов после первых пяти тестов при реализации метода кратных отказов сокращается в три раза, а при анализе полноты десяти тестов выигрыш во времени может достигнуть одного порядка.

В Таблицах 1 и 2 приведены некоторые экспериментальные данные. Испытания проводились для пяти тестов, проверяющих работу программы, исполнительный модуль которой занимает примерно 18 килобайт.

Таблица 1. Статистика прогонов тестов для одиночных отказов.

Номер теста	Кол-во отказов	Размер группы	Кол-во групп	Кол-во выявл. групп	Кол-во выявл. отказов	Всего прогонов
1	400	1	400	66	66	400
2	334	1	334	5	5	334
3	329	1	329	2	2	329
4	327	1	326	1	1	327
5	318	1	318	9	9	318
Итого						1708

Таблица 2. Статистика прогонов тестов для кратных отказов.

Номер теста	Кол-во отказов	Размер группы	Кол-во групп	Кол-во выявл. групп	Кол-во выявл. отказов	Всего прогонов
1	400	3	134	55	66	299
2	334	3	112	5	5	127
3	329	9	37	2	2	55
4	327	10	33	1	1	43
5	318	10	33	8	9	113
Итого						637

Средняя статистическая полнота тестов - 21%.

Увеличение выигрыша во времени от теста к тесту объясняется резким сокращением количества выявляемых отказов и тем, что в списке не выявленных отказов остаются отказы, относящиеся к подмножеству так называемых «трудно проверяемых».

Данные, помещенные в Таблицы 1 и 2, наглядно показывают выигрыш во времени при использовании метода кратных отказов по сравнению с методом внесения одиночных отказов примерно в 2,7 раза.

Кроме того, проведенные эксперименты продемонстрировали, что эффективность метода кратных отказов тем выше, чем меньше полнота тестов. При этом экспериментально установлено, что существует некоторый порог полноты анализируемого теста (30%-40%), при превышении которого использование кратных отказов нецелесообразно. Однако следует заметить, что, как правило, полнота отдельных тестов в классе рассматриваемых отказов не превышает 25%.

Полученные результаты показали, что отказы из случайной выборки, объединенные в группу, мало влияют друг на друга. Это позволяет предполагать, что подобный алгоритм будет особенно полезен при реализации мутационного подхода.

3 Совместный анализ интеграционных тестов объекта контроля и тестов его отдельных частей

Как правило, полнота анализируемых тестов не является удовлетворительной, и для выявления не обнаруженных отказов следует разрабатывать дополнительные тесты. Это весьма трудоёмкий процесс. Поэтому в данной работе предлагается, прежде чем приступить к синтезу дополнительных тестов, использовать для повышения полноты интеграционных тестов системы возможности тестов компонентов, и наоборот: использовать результаты оценки полноты интеграционных тестов для повышения качества проверки отдельных компонентов. Использование такого подхода сократит количество не выявляемых указанными тестами дефектов и, следовательно, сократит время для разработки дополнительных тестов.

Следует отметить, что в работе [15] предлагался оригинальный метод компонентного тестирования информационных систем, позволяющий на основе выделения наиболее часто встречающихся типов ошибок в программе выявлять их на уровне компонентов программ.

Суть предлагаемого в настоящей работе подхода заключается в следующем. Вначале осуществляется синтез тестов компонентов, поиск и исправление дефектов обнаруженных с их помощью в этих компонентах. После завершения данного этапа контроля аналогичные процедуры выполняются для системы с применением интеграционных тестов системы.

Теперь, после выявления и исправления всех обнаруженных тестами компонентов и тестами системы отказов можно приступить к оценке статистической полноты данных тестов в классе одиночных отказов (например, мутантов). Очевидно, что как дефекты, не выявляемые тестами системы, могут

быть выявлены тестами компонентов, так и невыявленные отказы компонентов могут быть обнаружены тестами системы.

Поэтому следует проанализировать возможность выявления непроверенных отказов интеграционных тестов системы тестами компонентов, которым эти отказы принадлежат. Для этого предлагается взять список непроверенных отказов после определения статистической полноты и путём последовательного внесения этих отказов в соответствующие компоненты определить, выявляют тесты компонентов эти отказы или нет.

Аналогичную процедуру можно применить для анализа списка непроверенных отказов компонентов. В этом случае непроверенные отказы компонентов последовательно вносятся в компонент, собирается версия системы и выполняются интеграционные тесты.

Как показали эксперименты, интеграционные тесты выявляют не менее 10% из невыявленных тестами компонентов отказов-мутантов.

Заключение

В работе рассматривался статистический метод оценки полноты тестов ПС и ДУ, обсуждались его достоинства и недостатки. Для модификации данного метода были предложены два подхода. Первый из них основывается на использовании кратных отказов вместо одиночных. Его применение позволит сократить время проверки полноты тестов в несколько раз. При этом показано, что в отличие от предложенного в [14] метода данный подход можно применять, начиная с первого теста. Второй подход использует пересечение множеств отказов контролируемого объекта и его отдельных компонентов. Благодаря этому удается повысить полноту, как интеграционных тестов, так и тестов отдельных компонентов ПС и ДУ и сократить время разработки дополнительных тестов.

В качестве дальнейших исследований в области оценки качества тестов нам представляется актуальным продолжать работы в области сокращения временных затрат. Одним из путей здесь видится сокращение выборки из множества отказов за счет некоторой потери точности как это предлагается, например, в [5,16].

Перспективным направлением является применение статистических методов для оценки качества и отказоустойчивости встроенных систем, когда имитация отказов сводится к внесению искусственных неисправностей в программное обеспечение этих систем [17].

Литература

1. Гробман Д.М. Программный контроль и диагностика неисправностей ЦВМ. // Диагностика неисправностей вычислительных машин. М.: Наука. 1965 г. С. 7 – 22.

2. Скобцов Ю. А., Сперанский Д. В., Скобцов В.Ю.Моделирование, тестирование и диагностика цифровых устройств. // Учебное пособие. М.: ИНТУИТ. 2012 г. 440 с.
3. Синицин С.В., Налютин Н.Ю. Верификация программного обеспечения. // Учебное пособие. М.: Бином. 2008 г. 307 с.
4. Seshu S. On an Improved Diagnosis Program // IEEE Trans. on Elec. Com-puters. 1965. Vol. 12, February, p. 76-79.
5. Корчемный Д.И. Оценка полноты тестирования методом последовательного анализа. // Вопросы радиоэлектроники. Сер. ЭВТ. 1980 г. Вып.17. С. 68 – 74.
6. Благодаренко А.В. Статистический и динамический анализ программного обеспечения без исходных текстов. // Проблемы информационной безопасности в системе высшей школы: труды XIV всероссийской научной конференции.–М.:МИФИ, 2007г. С. 33-34.
7. Mattias Bybro. A Mutation Testing Tool for Java Programs, Ett verktyg för mutationstestning av Javaprogram Examensarbete i Datalogi, 2003, 20p., http://www.nada.kth.se/~karlm/a_mutation_testing_tool_for_java.pdf.
8. Harman M., YueJia, Langdon W.B. A Manifesto for Higher Order Mutation Testing, Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on, 2010, 6-10 April, p.80-89.
9. Lorez C., RiesgoT.,De la Torre E.,Useda J. A method to perform error simulation in VHDL // Desing of Circuits and Integrated Systems Conference. Madrid (Spain), 1998. p. 495-500.
10. Offutt A.J., Untch R.H. Mutation 2000: Uniting the Orthogonal // Mutation Testing in the Twentieth and the Twenty First Centuries San Jose, 2000. p. 45-55.
11. Басок Б.М., Гречин А.А. О едином подходе при анализе тестов дискретных устройств и программ. // Вопросы радиоэлектроники. Сер. ЭВТ. 2010 г. Вып.3. С. 140 – 145.
12. Гробман Д.М. Статистический способ определения полноты тестов. // Тезисы докладов IV Всесоюзного совещания по технической диагностике. Черкассы. 1979 г. С. 9 – 11.
13. Сергеев Б.Г., Басок Б.М., Бродский М.А. Использование аппаратных средств моделирования в САПР СБИС. // Автоматизация логического проектирования средств вычислительной техники: труды первой международной конференции «САПР СВТ 89» - Л., 1989 г. С. 25-31.
14. Басок Б.М. Об одном методе оценки качества тестов дискретных компонентов вычислительных сетей. // Автоматика и вычислительная техника. 1985 г. №1. С. 56 – 58.
15. Кораблин Ю.П., Куликова Н.Л., Чумакова Е.В. Компонентное тестирование и его реализация. // Учебное пособие. М.: Союз, 2009 г. 23 с.
16. Рабинович Ю.Г. Ускорение статистического метода проверки полноты теста с использованием специализированных средств моделирования.// Труды Всесоюзного семинара «Специализированные процессоры в САПР». Тез.докл. - Москва, 1989. С. 17-18.
17. Thorhuus R. Software Fault Injection Testing. // Master of Science Thesis in Electronic System Design. Stockholm, Sweden: KungligaTekniskaHögskolan, 2000, 58 c.