Информатика

Грушин Анатолий Иванович
Кандидат технических наук, доцент Московского физико-технического института (МФТИ), ведущий научный сотрудник Института точной механики и вычислительной техники им. С.А. Лебедева РАН. Член IEEE, ACM и IEEE Computer Society. Автор около 50 печатных работ и 11 патентов, в том числе 2 патентов США.

Верификация в вычислительной технике

Согласно закону Мэрфи, если какая-то неприятность может случиться, то она случается. Человек может ошибиться, поэтому он обязательно ошибается. Ошибки случаются при проектировании аппаратуры, при написании программ, при изготовлении аппаратуры. Иногда они обходятся очень дорого.

Закон Мэрфи

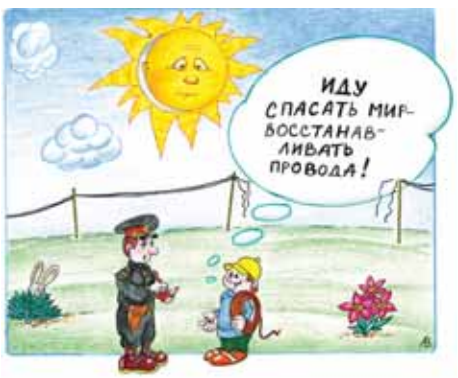
Капитан Эд Мэрфи служил в 1949 году на американской военно-воздушной базе, где исследовались причины аварий самолётов. Он был недоволен работой техников одной из лабораторий, утверждая, что если можно сделать что-то неправильно, то они только так и сделают. Непрерывные неполадки, вызванные действиями техников, на базе стали называть «законом Мэрфи».

С тех пор этот закон оброс множеством следствий для разных областей человеческой деятельности. Например, один из законов ненадежности Джилба гласит: число ошибок, которые нельзя обнаружить, бесконечно, в противовес числу ошибок, которые можно обнаружить, – оно конечно по определению.

В 1994 году профессор математики из Вирджинии при вычислении обратных величин простых чисел обнаружил, что микропроцессор Pentium в некоторых случаях неправильно делит числа с плавающей запятой. Представители фирмы Интел, производившей этот микропроцессор, заявили: «Это нельзя назвать ошибкой, – ведь даже если вы инженер, вы её не заметите. Если брать числа с плавающей запятой случай-

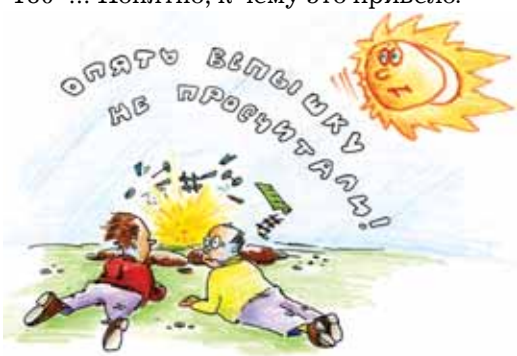
но, вероятность получить неточный результат $\sim 10^{-10}$ ».

Правда, Интел не объяснил, почему обычные пользователи будут брать числа случайно... Вскоре выяснилось, что Интел знал об ошибке, но скрыл её. Поднялась волна негодования, и в результате через месяц компания согласилась заменить микропроцессоры с неправильно спроектированным устройством деления. Этот случай не только обошёлся в 300 миллионов долларов, но и нанёс большой ущерб репутации фирмы. После этого стали шутить, что надпись на компьютере «Intel inside» (внутри микропроцессор Интел) – это не реклама, а предупреждение. Кстати, в 2006 году фирма Интел сменила этот рекламный слоган на новый – «Leap ahead» (прыжок вперёд).



В 2003 году в северо-восточных штатах США произошла крупнейшая в истории Америки авария в системе энергоснабжения. Без электричества остались 50 миллионов человек, экономические потери составили 6 миллиардов долларов. Одной из главных причин оказалась ошибка в программе, которая не смогла правильно обработать некоторое сочетание событий. Чтобы исправить эту ошибку, пришлось просмотреть несколько миллионов строк программы.

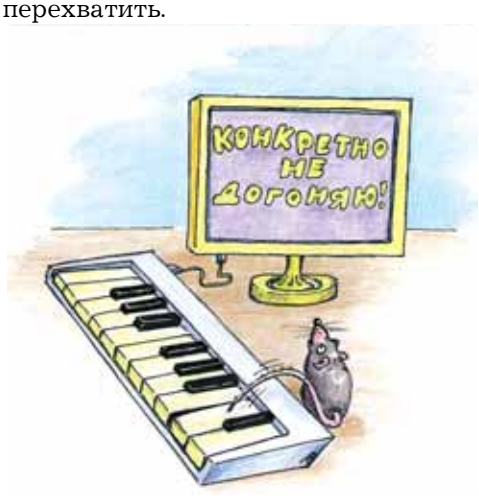
В военной области ошибки часто приводят к трагическим последствиям. В системе управления торпедой был предусмотрен её автоматический подрыв при отклонении от курса более чем на 90° для того, чтобы торпеда в случае неисправности не поразила корабль, с которого она пущена. Однажды торпеда при пуске застряла в торпедном аппарате, командир корабля решил вернуться в порт, чтобы устранить неисправность на базе. Он приказал повернуть на 180° ... Понятно, к чему это привело.



В 1983 году мир был на грани ядерной войны. Из-за ошибки советской системы раннего предупреждения блики солнца на облаках были восприняты как пуск американских ракет. На пульт системы поступило сообщение о запуске пяти американских ракет. К счастью, дежурный офицер усомнился в правильности работы системы, – он понимал, что первый ядерный удар не может наноситься таким небольшим количеством ракет. Похожая история произошла несколькими годами ранее с американской системой ПВО. За приближающиеся советские ракеты она приняла Луну...

В 1991 году во время Войны в заливе (США против Ирака) батарея американских зенитных ракет «Пэтриот» не смогла перехватить запу-

щенную иракцами ракету «Scud» советского производства. Ракета попала в казарму американских солдат, при этом погибло 28 человек. Причиной этого была погрешность вычисления времени. При вычислении нужно было умножать время, задаваемое тактовым генератором компьютера (оно измерялось в десятых долях секунды) на 1/10, но это десятичное число невозможно точно представить в двоичном виде. Для хранения этой константы использовался 24-разрядный регистр. Разница между точным значением 1/10 и её неточным двоичным представлением составляет в двоичном виде 0,0000000000000000000000011001100… или около 0,000000095 в десятичном виде. Компьютер был включён около 100 часов, за это время накопилась ошибка в измерении времени в 0,34 секунды. Скорость ракеты «Scud» составляла примерно 1700 м/сек, то есть за это время она прошла более 500 метров. Этого хватило для того, чтобы зенитные ракеты не смогли её перехватить.



Изредка ошибки могут обрадовать людей. В 1996 году из-за ошибки компьютера одного американского банка на счета 823 его клиентов было начислено почти по 925 миллионов долларов. В октябре 1995 года 200 тысяч государственных служащих во Франции получили зарплату дважды. В Германии после перехода страны от марки к евро банкоматы стали выдавать неограниченное количество денег при любом вводимом пароле.

Иногда ошибки вызывают улыбку. В сильно загруженной компьютерной системе постоянный поток процессов с высоким приоритетом не дает низкоприоритетным процессам доступа к ресурсам. Это заканчивается одним из двух способов. Либо процесс, в конце концов, получит долгожданный доступ, когда загрузка системы уменьшится, либо система «рухнет», и все незавершённые низкоприоритетные процессы будут потеряны. Рассказывают, что в 1973 году при выключении машины IBM 7094, которая работала в Массачусетском технологическом институте, был обнаружен незаконченный процесс с низким приоритетом, который был запущен в 1967 году…

Ещё один забавный случай. Кибернетическому цензору в Интернете не понравилась строка:

menu */#define

из-за запрещённого буквосочетания «nu…de» (по-английски – голый).

По некоторым данным, на 1000 строк обычной программы в среднем приходится 25 ошибок. В хорошей программе – 2 ошибки на 1000 строк. В космической отрасли вся аппаратура и программное обеспечение проверяются особенно тщательно, так как от их качества и надёжности зависит человеческая жизнь. Поэтому считается, что в программном обеспечении американского космического челнока содержится не более одной ошибки на 10000 строк. Тем не

менее, во время 10-дневного полёта американского космического корабля Аполло-14 было обнаружено 18 программных ошибок.

Banana software: Let the Software ripe at the customer!

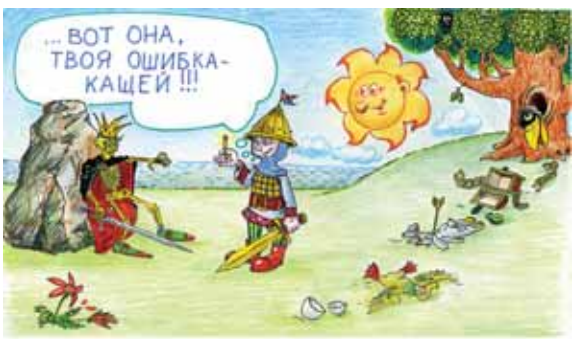
Плохо отлаженное программное обеспечение продаётся пользователю, там оно дозревает, как спеют зелёные бананы, пока их везут на теплоходе из жарких стран. Разница – бананы зреют сами собой, а на программное обеспечение приходится ставить patches – заплатки.

Разработчик программного обеспечения, когда ему указывают на ошибку в его продукте, отвечает: «Это не ошибка, это особенность» («This is not bug, this is feature»).

Приведённые выше примеры показывают важность верификации проектов. Слово *верификация* произошло от латинского *verus* – истинный и *facere* – делать. Вообще верификация означает подтверждение того, что описание проекта полностью соответствует спецификации (техническому заданию) проектируемой системы. Спецификация – это документ, подробно перечисляющий условия, которым должна соответствовать изготовляемая или проектируемая система.

Система может представлять собой аппаратуру или программные средства, а также некоторую их комбинацию. Аппаратура, особенно цифровая, в настоящее время проектируется с помощью языков описания аппаратуры, её проект таким образом и сам может считаться программой, поэтому в верификации аппаратных и программных систем много общего. Спецификация может представлять собой требования к функциональным возможностям, временным параметрам, потребляемой мощности, возможности использовать программные средства на различных вычислительных машинах, производительности, габаритам и т.д. Соответственно, верификации подвергаются различные аспекты проектируемой системы. Например, спецификация на разработку микропроцессора содержит систему команд, которую он должен выполнять, рабочую частоту, время выполнения команд в тактах, потребляемую мощность, используемую элементную базу, размер кристалла и тому подобное. Наиболее сложной и трудоёмкой является *функциональная* верификация, которая и будет рассматриваться далее.

Функциональная верификация проекта – это процесс доказательства того, что он функционирует согласно своей спецификации. Например, при разработке микропроцессора функциональная верификация заключается в проверке правильности выполнения команд, входящих в систему команд. Логические ошибки



могут задержать появление системы (аппаратной или программной) на

рынке или привести к отказу уже находящейся в эксплуатации.

Верификация начинается с разработки тест-плана, в котором описывается, как предполагается тестировать, какими инструментальными средствами, какие тесты использовать и так далее. Затем создаётся среда для верификации, в которую будет помещена модель проектируемой системы. После стыковки (совместной отладки) среды для верификации и модели системы прогоняются тесты, проверяющие систему, при этом оценивается качество тестов.

Моделирование и формальная верификация являются основными методами функциональной верификации.

Моделирование подразумевает подачу определённых входных воздействий на модель проектируемой аппаратной системы, наблюдение результатов этих воздействий на систему и принятие решения, верны ли результаты. Термин моделирование применяется к аппаратным системам, а термин тестирование относится и к программным, и к аппаратным системам.

Более абстрактная модель описывает общее поведение аппаратной системы, но оставляет без внимания некоторые внутренние особенности. Менее абстрактная модель даёт больше подробностей и ближе к действительной реализации системы. Обычно более абстрактная модель пишется быстрее, и она моделируется быстрее, а более подробная модель используется для синтеза реальной системы. В сложных проектах обычно начинают с более абстрактной модели, описывающей алгоритмические и архитектурные аспекты, а затем создают более подробную модель, содержащую информацию, необходимую для синтеза. Более абстрактные модели легче читать, понимать и отлаживать.

Нет одного, лучшего, метода генерации входных воздействий и анализа результатов. Один из самых простых методов генерации теста – точно указать, какие величины нужно посылать на каждый вход системы, это даёт так называемые направленные тесты.

В случайном тесте входные воздействия задаются случайным образом с использованием генератора псевдослучайных чисел. Далеко не всегда случайный тест хорошо проверяет. Так, с его помощью практически невозможно проверить, как работает нормализатор результата устройства, складывающего числа с плавающей запятой. Ведь для его проверки нужно, чтобы порядки слагаемых были одинаковые, знаки чисел были разные, и совпало некоторое количество старших разрядов мантисс. В этом случае при сложении мантисс в старших разрядах суммы получатся нули, то есть, ситуация, требующая работы нормализатора. В случайном тесте выполнение перечисленных выше условий очень маловероятно. Например, для формата double extended, используемого в микропроцессорах Pentium, вероятность того, что при случайных операндах сумма мантисс будет иметь 10 нулей в старших разрядах, $\sim 10^{-8}$. Если в случайном тесте есть некоторый уровень управления случайностью для того, чтобы тест выполнял желаемые операции, то это направленный случайный тест.

Чтобы проверить, что система работает, как планировалось, нужно следить за реакцией системы на входное воздействие. Это можно делать вручную или сравнивать ре-

зультат моделирования с результатом, получаемым на другой модели.

64-разрядный сумматор для полной проверки требует подачи $2^{64} \times 2^{64}$ различных комбинаций входных сигналов, что составляет более 10^{38} комбинаций. Пусть время моделирования одной комбинации слагаемых составляет 1 наносекунду (10^{-9} с), тогда время перебора всех возможных входных комбинаций составит 10^{29} с, а это примерно 3×10^{21} лет! Возраст существования Земли во много раз меньше.

Наличие внутри проверяемого узла элементов памяти значительно увеличивает количество входных комбинаций, необходимых для полного перебора, так как выход схемы, содержащей элементы памяти, зависит не только от входных сигналов, но и от того, что хранят элементы памяти, то есть от их состояния.

Поэтому входные воздействия для проверки строят не из соображений проверки правильности работы схемы на всем множестве входных наборов, а из соображений активизации путей для каждой из неисправностей, в результате чего число входных наборов, на которых следует верифицировать схему, оказывается приемлемым для моделирования.

При тестировании возможны два подхода – метод «чёрного ящика» и метод «прозрачного ящика». В первом случае при разработке теста не используются знания о внутренней организации системы, во втором – используются.

Существует несколько методов оценки качества функциональной верификации: покрытие программы, функциональное покрытие, покрытие путём вставления ошибок, изучение проекта и тест-планов.

Покрытие программы – считают, какие строки описания проекта исполнялись при тестировании.

Функциональное покрытие – проверяется, что все известные допустимые состояния блока программы были исполнены.

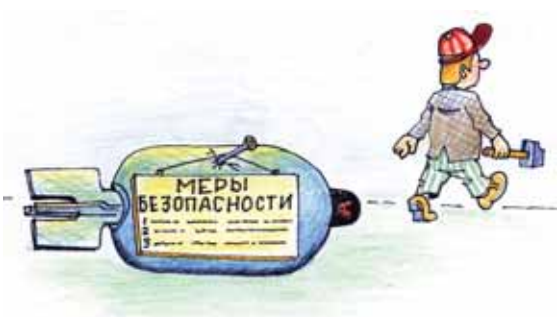
Покрытие путём вставления ошибок – вставляется ошибка, и смотрят, ловит ли её тест.

Изучение проекта и тест-планов – проверяется, все ли разделы спецификации тестируются.

Моделирование не позволяет узнать, выявлены ли в системе все ошибки.

Методы формальной верификации доказывают, что система не содержит ошибок, они относятся к основанным на математике способам спецификации, разработки и верификации программных и аппаратных систем. В формальной верификации используется как ручное, так и автоматическое доказательство теорем. Оба метода требуют участия человека. Одним из методов формальной верификации является доказательство эквивалентности. Например, модель быстрого, но сложного сумматора можно сравнивать на эквивалентность с простым, но медленным сумматором, модель которого заведомо правильна.

Формальная верификация обеспечивает исчерпывающий анализ всех возможных вариантов поведения системы.



Отладка – это процесс тестирования и устранения ошибок в программе или в аппаратуре с целью заставить её вести себя, как предписано спецификацией.

Знаменитый учёный в области программирования Эдсгер Дийкстра сказал: «Если отладка – это процесс удаления ошибок, то программирование – это процесс внесения ошибок».

Основные шаги отладки: узнать, что ошибка существует; локализовать источник ошибки; установить причину ошибки; определить, как её исправить; исправить ошибку и проверить.

Методы моделирования (тестирования) эффективны на ранних стадиях отладки, когда в проектируемой системе ещё очень много ошибок. По мере отладки в системе остаётся меньше ошибок, и результативность этих методов снижается, при этом резко возрастает время, которое требуется для обнаружения более сложных ошибок.

Для установления причины ошибки нужно хорошо понимать систему.

При отладке полезно иметь регрессионные тесты, которые повторяются при внесении изменений в систему. Они позволяют выявить ошибки, которые случаются из-за некорректных изменений, когда перестаёт работать то, что раньше функционировало правильно.

Для верификации аппаратуры часто используется физическая модель (прототип) на программируемых логических интегральных схемах (ПЛИС). Это позволяет значительно увеличить скорость тестирования и, тем самым, увеличить количество прогоняемых тестов, то есть улучшить качество тестирования.

Тестирование программного обеспечения – процесс определения правильности, полноты и качества разработанного программного обеспечения.

Альфа-тестирование – это проверка программного обеспечения самими разработчиками.

Бета-тестирование – передача программного продукта группе людей для опытного использования (ошибок должно быть уже немного). Иногда бета-версии открывают для широкой публики, чтобы выявить как можно больше ошибок.

Некоторые циничные программисты называют выпуск программного продукта на рынок гамма-тестированием. Причина этого в том, что почти в любом программном продукте пользователи, в конце концов, обнаруживают ошибки.

При проектировании аппаратуры функциональная верификация занимает более половины людских ресурсов, времени и стоимости. Верификация программных систем также требует больших усилий, особенно в таких применениях, как управление атомным реактором, противоракетная оборона и так далее.