Автоматизация модульного тестирования с Atollic TrueVERIFIER

для повышения качества встраиваемых приложений

Анна СЕРГЕЕВА
annserge@rambler.ru

Нередко случается, что встраиваемые системы поступают на рынок без достаточного тестирования. И это неудивительно, ведь весьма большой процент разработчиков таких систем не относится всерьез к внедрению и использованию каких-либо инструментов для проведения тестирования. А ведь подобное пренебрежение самым серьезным образом сказывается на качестве выпускаемой продукции, и, разумеется, не в лучшую сторону.

Проблема еще больше усугубляется тем, что средств целевого тестирования встраиваемых программных приложений на рынке представлено не так уж много.

В статье показано, какую поддержку специалистам может оказать применение современных инструментов автоматизированного целевого тестирования. Почему эта область деятельности столь важна при создании встраиваемых систем. Как подобные инструменты способны сократить количество часов необходимой инженерной деятельности при одновременном увеличении полноты тестового покрытия и повышении качества программного обеспечения.

Введение

В большинстве случаев выпуск продукции со скрытыми проблемами и неисправленными ошибками приводит к необходимости поставки клиентам дополнительных обновлений, отзыва и ремонта неисправного оборудования и т. п. И разумеется, все затраты на подобную деятельность обойдутся производителю очень недешево. Причем это не только скажется на материальном состоянии компании, но и ударит по репутации, а значит, все меньшее число клиентов сохранит доверие к компании и пожелает продолжать сотрудничество.

Но, несмотря на подобные риски, многие программные приложения, предназначенные для встраиваемых систем, выпускаются без каких-либо тестовых испытаний. Хотя известно, что тестирование способствует если не полному устранению, то по крайней мере значительному снижению подобных проблем.

И все же немало разработчиков встраиваемых систем выбрали другой путь и предпочли внедрение и использование автоматизированного целевого тестирования программ. Перед ними стоит непростая задача — найти инструменты, которые хорошо подходят для интегрирования со встроенной средой разработки. Данные требования обусловлены тем, что инструменты с подобным уровнем интеграции обеспечивают более эффективное тестирование и исправление обнаруженных ошибок [1].

Большинство существующих программных средств тестирования позволяет выполнять тесты лишь на платформе персональных компьютеров (ПК). Это ограничивает их применение при тестировании встроенных приложений, поскольку фактические системы будут иметь совершенно иные аппаратные интерфейсы, другие проблемы синхронизации, ограничения памяти, особенности встроенного ассемблера, несовместимость файлов конфигурации и прочие существенные различия.

Кроме того, исходный код, совместимый со стандартом ANSI-C, часто ведет себя поразному при обработке различными компиляторами или при запуске на процессорах с разной архитектурой. Типичными примерами являются формирование пакетов и упорядочивание структурных единиц, установка разных размеров по умолчанию для определенных типов данных (например, 8 или 32 бита для данных типа char) и т. д.

Перекомпиляция встраиваемого кода для запуска на ПК часто влияет на его поведение во время исполнения, что может усложнить понимание кода или привести к получению ложных результатов. Один и тот же тест, который успешно проходит на ПК, вполне способен завершиться неудачей при запуске на отладочной плате.

Следовательно, особую важность имеет прогонка как можно большего числа программных тестов именно на фактическом встраиваемом оборудовании и за счет этого минимизация вероятных расхождений между платформами. Именно поэтому инструменты, предназначенные для целевого тестирования, обладают значительным преимуществом перед теми, которые ориентированы на тестовую симуляцию приложений на каких-либо внешних платформах (таких как ПК и другие).

Важность раннего обнаружения ошибок

Нахождение и исправление ошибок — это неизбежные составляющие любого процесса разработки программного обеспечения. Но применительно к созданию встраиваемых приложений данный процесс приобретает особое значение.

Специалисты знают, что поиск ошибок в начале цикла гораздо дешевле, чем более позднее их обнаружение [2]. Однако при традиционных стратегиях тестирования и отладки многие ошибки могут оставаться скрытыми слишком долго и быть обнаружены уже на таких поздних стадиях разработки и тестирования, когда стоимость их исправления окажется неоправданно высокой. Вот почему команды разработчиков должны стремиться к нахождению и исправлению ошибок уже на первых этапах цикла.

Совершенно понято, что грамотный руководитель проекта поддержит такую стра-

тегию и будет больше заботиться о качестве программного обеспечения на ранних стадиях процесса разработки. Ведь руководство прекрасно осознает, что любые ошибки, найденные и исправленные в самом начале, обойдутся дешевле, чем поиск и решение проблем во время тестирования или оказания технической поддержки.

Применение специализированных инструментов значительно облегчает создание тестовой среды уже на ранних стадиях процесса разработки встраиваемых приложений, а также упрощает проведение тестирования и повышает тестовое покрытие кода. Кроме того, становится намного легче выполнять и регрессионные тесты, которые гарантируют, что внесенные изменения не приведут к появлению новых ошибок в существующем коде.

Модульное тестирование

Для большинства встраиваемых приложений общепринятые методики тестирования программных приложений не подходят.

Тем не менее все больше специалистов обращается к методологии модульного тестирования (unit testing), находя ее весьма эффективной для тестирования встраиваемых приложений.

Модульные тесты — это тесты, которые направлены на конкретные С-функции и вызывают их с различными комбинациями значений входных параметров, что приводит к различным путям выполнения кода этих функций.

Сложности применения к встраиваемым приложениям

Разумеется, написание модульных тестов вручную — очень трудоемкий, дорогостоящий и однообразный процесс. К тому же велика вероятность, что многие из возможных путей выполнения программы останутся непроверенными.

Для того чтобы написанные тесты приводили как можно к большему числу важных путей выполнения кода функции, требуется проведение тщательного анализа того, какие входные данные понадобятся для направления кода по разным путям выполнения. Это чрезвычайно трудоемкая и сложная, а порой и невыполнимая работа. А гарантированный результат может быть обеспечен разве что для самых тривиальных функций.

Здесь напрашивается вывод, что просто необходимы специализированные инструменты, которые выполняют всю эту работу автоматически.

Еще одна проблема, с которой сталкиваются команды разработчиков,— поддержание синхронизации модульных тестов с исходным тестируемым кодом, находящимся в разработке и постоянно изменяющимся.

Часто проблема усугубляется и очень плотным графиком работы команды, и нередко по мере развития проекта сроки все больше сжимаются. Так что если утрачивается синхронизация исходного кода и его модульных тестов, то последние теряют свою пользу, особенно в моменты, когда они наиболее необходимы.

И конечно, следует отдельно отметить, что сама специфика встраиваемых систем усложняет их тестирование.

Ведь изначально все тестирование проводится в удаленном режиме. А значит, наиболее эффективными модульными тестами будут те, что встроены в само приложение и запущены на целевом оборудовании в такой среде приложения, в которой действительно их предстоит использовать. И эффект здесь окажется значительно выше, чем при прогоне тестов на вспомогательных платформах, способных только работать в режиме симуляции.

Выбор стратегии

Существует широкий ряд разнообразных инструментов, успешно применяемых для модульного тестирования приложений для ПК. Однако от них мало пользы разработчикам встраиваемых приложений. Ведь с помощью таких инструментов редко удается выполнить компиляцию, загрузку и проигрывание наборов тестовых сценариев на встраиваемом оборудовании.

Однако есть и специализированные инструменты, которые способны создавать модульные тесты автоматически, внедряют их во встраиваемые приложения и позволяют запускать их на целевом оборудовании в режиме JTAG-отладки. Применение таких инструментов приведет к оптимальному расходованию рабочего времени и максимальной производительности.

Но еще более эффективно использование полностью встраиваемой системы автоматизации тестирования, интегрированной в среду разработки (IDE). Это позволяет легко поддерживать синхронизацию исходного кода и модульных тестов.

Ранее в индустрии встраиваемых систем подобные инструменты широкого распространения не имели. Но в последнее время появляются все новые специализированные средства модульного тестирования, среди которых можно назвать Atollic TrueVERIFIER.

Возможности Atollic TrueVERIFIER

Данный инструмент модульного тестирования в полной мере соответствует всем строгим требованиям, предъявляемым к тестированию встраиваемых систем. Он поддерживает такие функциональные возможности, которые позволяют разработчикам обеспечивать высокий уровень целевого автоматизированного тестирования [3].

Среди ключевых возможностей Atollic TrueVERIFIER:

- автоматическое генерирование модульных тестов;
- ручное редактирование созданных тестовых наборов;
- проведение функционального тестирования;
- проведение тестирования совместимости (интеграционное тестирование);
- автоматическое исполнение тестовых наборов на целевом отладочном оборудовании;
- оценка качества проведенного тестирования.

Автоматическое генерирование модульных тестов

Новое поколение профессиональных инструментов автоматизации тестирования для встраиваемых систем способно выполнять анализ исходного кода, а затем генерировать и реализовать любые целесообразные тесты на целевых отладочных платах. И все это самостоятельно, в автоматическом режиме. В итоге необходимость в ручном инженерном труде, по сути, отпадает.

Современные инструменты могут быть задействованы как для проверки поведения отдельных функций (для функционального тестирования), так и для проигрывания тестовых сценариев при исследовании совместной работы функций в типовых случаях их использования (речь здесь идет об интеграционном тестировании).

Функциональное тестирование

Atollic TrueVERIFIER выполняет синтаксический анализ (парсинг) исходного кода приложения и внедряет в него автоматически генерируемый набор тестов. Эти тесты выполняют многократный вызов исследуемых функций с разными комбинациями входных параметров, что обеспечивает прохождение кода наибольшим числом различных путей исполнения.

На рис. 1 приведен показательный пример тестирования тривиальной С-функции путем ее многократного вызова с разными входными параметрами. При этом инструмент тестирования выбирает и генерирует необходимые значения входных параметров (тестовых данных или тестового вектора) самостоятельно, то есть автоматически.

Как правило, в тестирование попадают не только граничные значения (минимальные, максимальные и приближенные к 0), но и любые другие значения, которые будут оказывать влияние на ход исполнения функции.

Для сохранения полной свободы действий и для обработки дополнительных важных случаев наряду с работой в автоматическом режиме Atollic TrueVERIFIER позволяет редактировать или создавать тесты вручную, путем добавления пользовательских наборов значений к сгенерированным наборам входных параметров.

В дополнение к автоматическому обнаружению, какие из значений входных параметров влияют на ход исполнения функций,

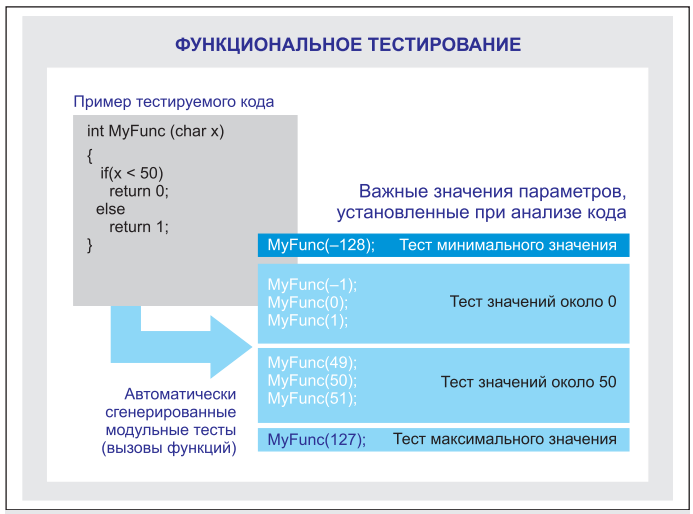

Рис. 1. Автоматическое генерирование модульных тестов

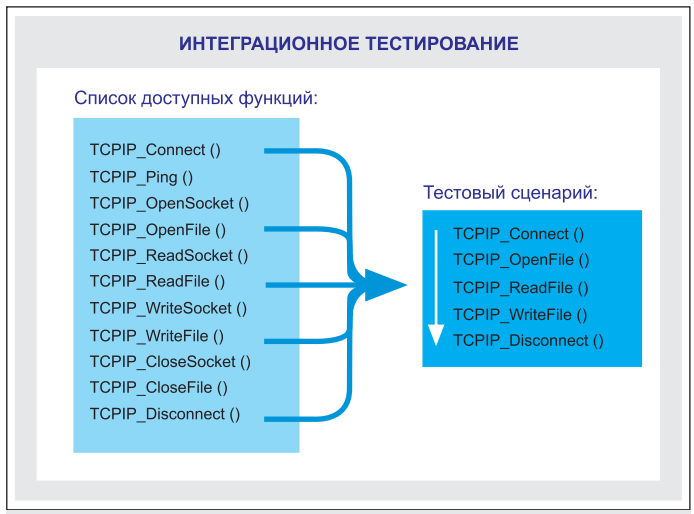

Рис. 2. Создание сценариев для интеграционного тестирования

Atollic TrueVERIFIER способен обнаруживать и тестировать и другие входные зависимости, такие как влияние на работу функций используемых глобальных переменных.

Интеграционное тестирование

Помимо исследования поведения каждой отдельно взятой функции (то есть функционального тестирования, о котором речь шла в предыдущем разделе), часто особый интерес представляет тестирование совместной работы функций, или интеграционное тестирование. Это исследование поведения исполняемого кода с более общей точки зрения. Базируется оно на составлении набора тестовых сценариев, в рамках которых тестируется совместная работа функций в штатном режиме и по назначению решаемых задач. Atollic TrueVERIFIER поддерживает и этот вид тестирования (рис. 2).

Так что с его помощью разработчики, выполняющие модульное тестирование, могут легко создавать тестовые сценарии, вызывающие определенный набор функций в заданном порядке и с предопределенными значениями входных параметров.

Редактирование сгенерированных тестовых наборов

В результате автоматического создания набора тестов, предназначенных для функционального или интеграционного тестирования, Atollic TrueVERIFIER располагает набором тестовых векторов (они создаются автоматически в ходе работы инструмента). Эти векторы содержат набор тестовых случаев (по сути, набор вызовов функций) со значениями входных параметров, которые обеспечат наибольшее число различных путей исполнения для исследуемых функций.

Таким образом, специалисты, выполняющие модульное тестирование, получают целый ряд возможностей дорабатывать тесты вручную. С помощью редактора тестовой конфигурации (рис. 3) Atollic TrueVERIFIER позволяет:

- редактировать наборы тестовых случаев или их входных параметров;
- добавлять проверки возвращаемого кода;
- добавлять код инициализации или завершения работы;
- применять пользовательские способы обнаружения неисправностей произвольной сложности.

Панель **Project Explorer** предназначена для удобной навигации по рабочим файлам. В дереве **Test Suite List** можно перейти к тому или иному тестовому набору, который необходимо модифицировать. На панели **Unit Test** приведен перечень доступных модульных тестов в наборе. В нижней панели отображаются результаты проведения тестирования в нескольких представлениях: **Test cases** (успех/ошибка выполнения конкретных тестовых случаев с конкретными входными данными), **Global Variables** (исследование влияния на работу функций используемых глобальных переменных), Coverage (измерение тестового покрытия кода).

В редакторе доступно добавление, редактирование и удаление тестов — все это можно делать по мере необходимости.

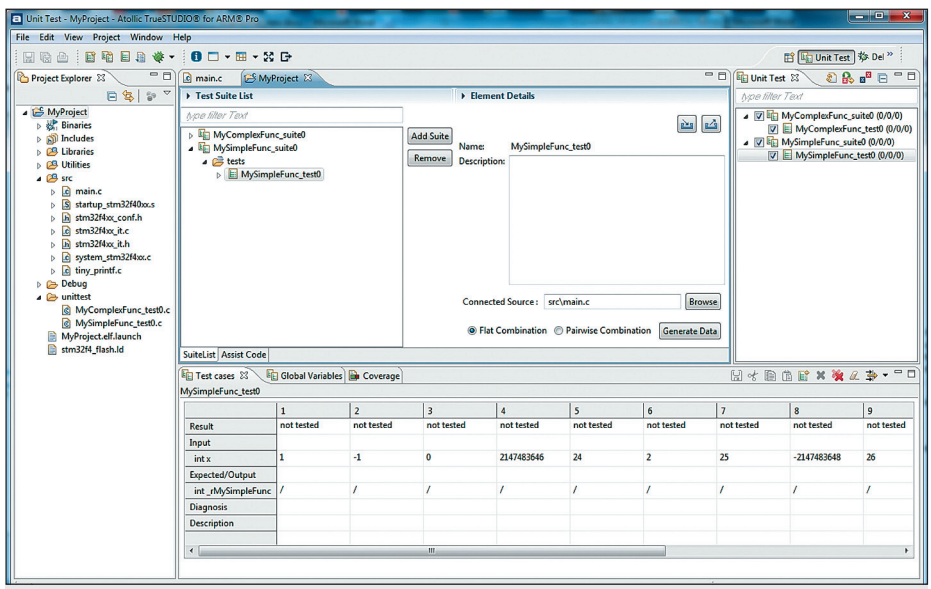

Рис. 3. Редактор тестовой конфигурации для модульного тестирования Atollic TrueVERIFIER

Дополнительные возможности

Тестовые векторы могут быть экспортированы для переноса во внешние сторонние инструменты (например, в Microsoft Excel) или для обратного импорта в данную систему тестирования.

Следует отметить еще одну удобную особенность Atollic TrueVERIFIER. Он предусматривает автоматическое выполнение проверки возвращаемого кода: для заданных входных параметров достаточно просто указывать ожидаемый возвращаемый код в каждом тестовом случае.

Можно еще более широко использовать данный инструмент. Разработчики, выполняющие модульное тестирование, могут

определять код инициализации и завершения работы аппаратной или программной среды, который будет задействован еще до запуска тестовых случаев или целых тестовых наборов.

Все это обеспечивает простоту применения Atollic TrueVERIFIER для исследования типовых тестовых ситуаций, а также позволяет создавать и исполнять и более сложные тестовые сценарии.

Автоматическое выполнение целевых тестов

После создания (и возможного редактирования) тестовых наборов они должны быть скомпилированы, привязаны к целевой отладочной системе и выполнены на ней.

Большинство недорогих инструментов модульного тестирования запускают тесты только на платформе ПК, работающих под ОС Windows. И отсутствие возможности интегрировать эти инструменты в среды разработки встраиваемых систем не позволяет выполнять тесты на отладочных платах. Подобные типы инструментов не отвечают высоким требованиям профессионального тестирования встраиваемых систем, что приводит к их неэффективности в данной сфере.

Совсем другое дело инструменты автоматизации тестирования, которые являются составным компонентом (а не дополнительной надстройкой) самого пакета инструментов разработчика. Они гарантируют наиболее эффективный способ синхронизации создаваемого кода и тестов, поскольку тесты регулярно обновляются в соответствии со всеми изменениями в коде. А это дает разработчикам больше шансов обнаружить ошибки на ранних стадиях жизненного цикла ПО, прежде чем стоимость исправления ошибок и решения проблем станет чрезмерно высокой.

К числу таких инструментов относится и Atollic TrueVERIFIER. Он полностью интегрируется в Atollic TrueSTUDIO, профессиональную среду разработки, компиляции и отладки кода встраиваемых приложений на языке C/C++ [4]. Это помогает легко синхронизировать разработку кода и тестов и пользоваться преимуществами интеграции со вспомогательными инструментами, такими как JTAG-отладчики.

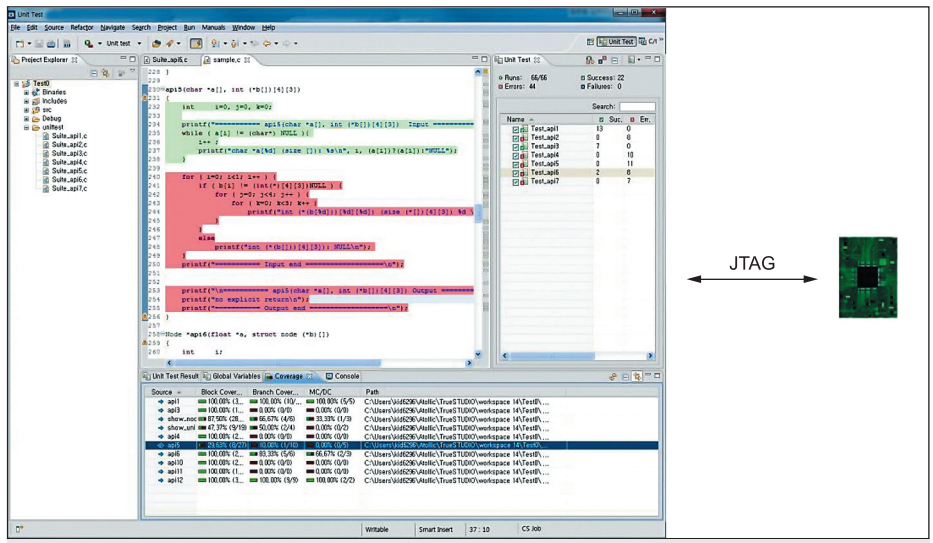

Рис. 4. Взаимодействие инструмента автоматизации тестирования встраиваемых систем с целевой платформой

Как только модульные тесты сгенерированы и помещены в исходный С-код, они автоматически компилируются с помощью интегрированных инструментов сборки встраиваемых приложений, а затем загружаются на целевую отладочную плату для использования в режиме JTAG-отладки (рис. 4). Далее осуществляется выполнение тестов на отладочной плате и динамический анализ прохождения тестов для измерения достигнутого покрытия кода.

После выполнения тестов на целевой отладочной плате информация о результатах тестирования и о покрытии кода загружается обратно в среду разработки (рис. 5).

В примере показан неуспех обработки функции ***_rMySimpleFunc*** со входным параметром int x = 24. Результат подсвечен на интерфейсе красным цветом. Как видно на рисунке, ожидаемое (Expected) возвращаемое значение было 1, а в действительности (Output) функция вернула 0. Соответственно, в графе результата (Result) указан неуспех (failure) обработки данного тестового случая.

Atollic TrueVERIFIER поддерживает измерение покрытия кода на достаточно высоком уровне — MC/DC. (Это уровень качества тестирования, предъявляемый к ПО для систем управления полетами, подробнее рассматривается в следующем разделе.)

Итак, по сути, профессиональные инструменты автоматизации тестирования встраиваемых систем, такие как Atollic TrueVERIFIER, автоматизируют следующие задачи:

- Анализ исходного кода конкретных С-функций для выяснения, какие комбинации значений входных параметров повлияют на ход выполнения этих функций.
- Генерирование тестовых наборов (в исходном С-коде), которые многократно вызывают исследуемые функции с различными комбинациями значений входных параметров, создавая различные пути выполнения этих функций.
- Компиляция исходного кода тестовых наборов и привязка к целевой системе.
- Загрузка тестового набора на целевую отладочную плату в формате двоичного кода для использования в режиме JTAG-отладки.
- Выполнение тестовых наборов на целевой отладочной плате с отслеживанием хода выполнения, что позволяет проводить расширенный анализ покрытия кода для измерения достигнутого качества тестирования.
- Загрузка и визуализация результатов тестирования и достигнутого покрытия кода.

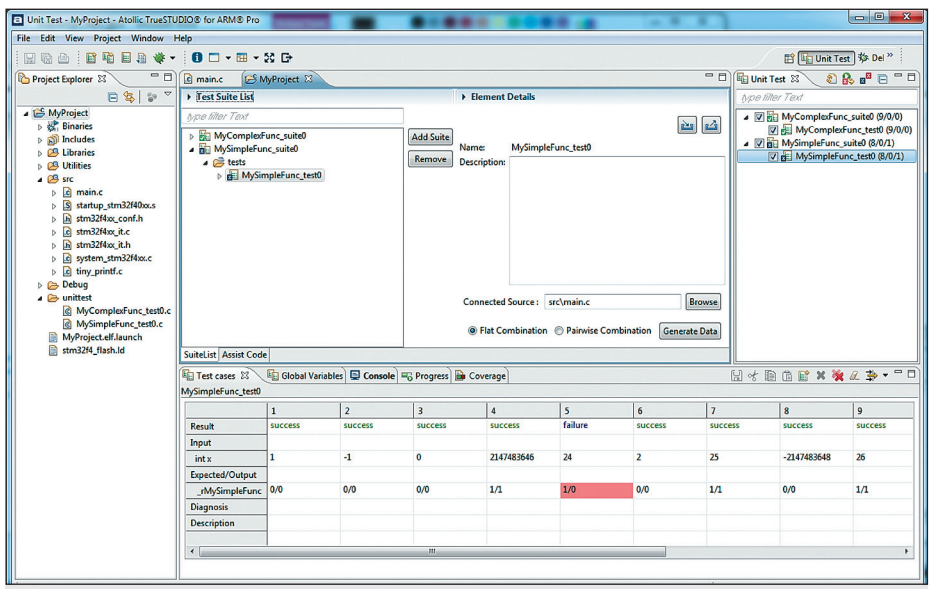

Рис. 5. Данные о результатах тестирования и достигнутом покрытии кода загружаются в среду разработки

Оценка качества проведенного тестирования

Поскольку создание наборов тестов для модульного тестирования и их выполнение являются автоматическими, нужно еще проверить, насколько хорошо на самом деле было выполнено такое тестирование программного кода и в какой степени можно доверять его результатам.

Лучший способ оценки качества модульного тестирования — изучение его результатов. И здесь необходимо измерение покрытия кода.

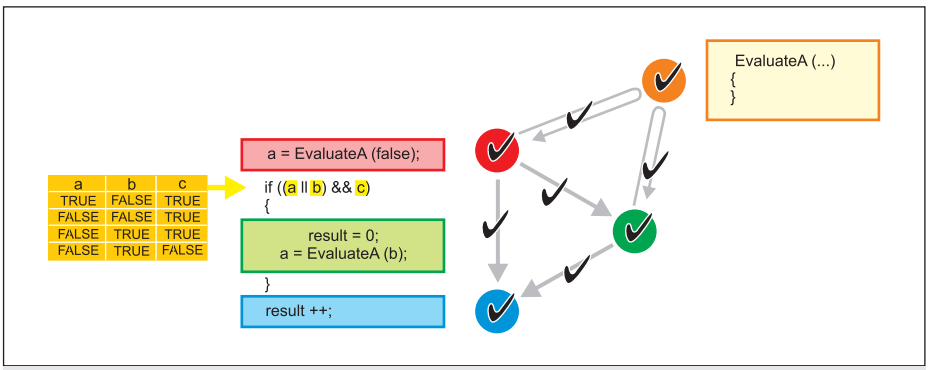

Рис. 6. Измерение качества тестирования с применением анализа покрытия кода

Анализ покрытия кода

Анализ тестового покрытия кода выполняется с помощью динамического анализа потока исполнения кода. Обычно в ходе анализа проводится исследование, какие части кода были протестированы, и на основании этих данных удается измерить качество тестирования [5], [6].

Существует большое количество различных методов анализа покрытия кода, от самых простых до достаточно жестких по требованиям к оценке. Среди них:

- метод покрытия по строкам или блокам кода;
- метод покрытия функций кода;
- метод покрытия вызовов функций;
- метод покрытия ветвей кода;
- модифицированный метод покрытия по ветвям/условиям (Modified Condition/Decision Coverage или MC/DC).

Разумеется, это достаточно формальная классификация видов анализа покрытия кода. А для измерения качества тестирования критичного по безопасности программного обеспечения применяются более сложные комбинированные методы. Ведь практика показывает, что даже самые простые участки кода очень сложно протестировать тщательно. И чтобы обеспечить достаточно точное тестирование, все блоки кода должны быть исполнены, все ветви должны быть пройдены и все вложенные выражения всех ветвей условных операторов должны быть проверены.

Только так можно гарантировать, что пройдены все возможные пути исполнения исследуемого кода. А для критичных по безопасности встраиваемых систем это имеет решающее, а в ряде случаев даже жизненно важное значение.

В качестве такого достаточно серьезного примера можно привести стандарт RTCA DO-178B. Он применяется для разработки критичного по безопасности ПО для летательных аппаратов. Здесь требуется выполнять тестирование по модифицированному методу MC/DC применительно ко всем наиболее критичным компонентам авиационного бортового ПО, где программная ошибка может привести к катастрофическим последствиям вплоть до потери воздушного судна и человеческих жизней.

Стоимость ошибок, и почему так важно их избегать с самого начала

Самое интересное, что приведенный подход применим отнюдь не только к авиационно-космической отрасли.

Он пригодится достаточно широкому кругу разработчиков встраиваемых приложений в разных отраслях, и они смогут извлечь выгоду из практики более тщательного тестирования и оценки результатов.

К примеру, для компаний, поставляющих продукцию в крупных объемах, с дефектами от программных ошибок столкнется достаточно большое число клиентов и, соответственно, исправление этих проблем обойдется гораздо дороже.

Еще один довод. Некоторые программные продукты устанавливаются на достаточно удаленном объекте или имеют особый режим использования (критичный к остановке и отключению). Это создает определенные сложности в организации и проведении их обновлений, а также сказывается на повышении стоимости сопроводительных работ. А их можно было бы избежать (или значительно минимизировать), если бы в процессе разработки приложения проводилось более тщательное тестирование.

Во всех случаях, когда поставщики стремятся сохранить свое доброе имя, любые дефекты предлагаемого ими программного обеспечения могут дорого обойтись компании, снижая репутацию в отрасли, степень доверия клиентов к компании и долю на рынке.

Именно поэтому всегда, когда только возможно, необходимо применять анализ покрытия кода для выяснения, достаточно ли хорошо протестирован код разработанного ПО. И делать это нужно именно до поставки готовых программных продуктов клиентам. И чем раньше, тем лучше.

На рис. 6 проиллюстрировано, как Atollic TrueVERIFIER отслеживает поток выполнения функций во время прогона набора тестов.

В приведенном примере выражение A проверяется таким образом, что учитываются все результаты обработки вложенных выражений (a, b, c) во всех ветвях кода. Другими словами, данное выражение должно быть протестировано по меньшей мере 4 раза.

Такая, даже относительно простая конструкция кода приводит к возникновению множества возможных путей исполнения.

А в реальных ситуациях часто используются гораздо более сложные конструкции кода, что приводит к усложнению тестирования и покрытия кода.

Atollic TrueVERIFIER измеряет покрытие кода с применением метода MC/DC, что обеспечивает уровень качества тестирования, требуемый для критичного по безопасности авиационного бортового ПО.

Для каждой тестируемой функции Atollic TrueVERIFIER проверяет, все ли блоки и ветви кода выполнены. Также он проверяет, все ли функции вызваны, все ли вызовы функций реализованы. И все ли вложенные выражения в составных условных операторах были обработаны.

Эти данные обрабатываются, сопоставляются и отображаются в результирующем

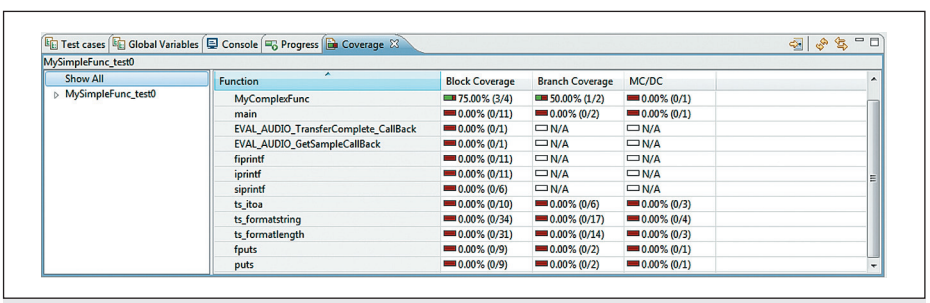

Рис. 7. Atollic TrueVERIFIER также измеряет уровень качества проведенного тестирования

отчете, что позволяет легко и удобно выполнять проверку эффективности проведенного тестирования (рис. 7).

В нижней панели программы на табе Coverage доступен просмотр результатов покрытия как для всех функций (выбор **Show All**), так и для конкретных сгенерированных наборов и функций, доступных в списке (в примере сгенерирован единственный набор ***MySimpleFunc_test0***).

В таблице перечислены функции выбранных наборов (Function), результаты оценки покрытия по блокам кода (Block Coverage), покрытия ветвей кода (Branch Coverage) и оценка покрытия по методу MC/DC. Значения приводятся в процентном и абсолютном соотношении. Для удобства восприятия дополнительно реализована цветовая индикация: зеленый (доля объема кода, покрытого модульными тестами), красный (для кода, не охваченного тестированием), белый (для случаев, когда оценка покрытия не может быть выполнена).

Помимо того что Atollic TrueVERIFIER способен создавать модульные тесты в автоматическом режиме, у него есть и другие достоинства.

В частности, он бесшовно интегрируется с другими инструментами в среде разработки. А это позволяет загружать инструментованный (снабженный модульными тестами) код на целевую платформу, запускать на ней наборы тестов, собирать результаты испытаний и выполнять оценку качества тестирования для дальнейшего анализа.

Такой подход дает разработчикам встраиваемых приложений очень высокую уверенность в фактическом тестовом покрытии и достигнутом качестве проведенного тестирования.

Заключение

В большинстве случаев разработчики встраиваемых приложений проводят тестирование на платформах, которые способны только симулировать, но не полностью воспроизводить работу тестируемого устройства. А это значит, что приходится дополнительно затрачивать значительное время на анализ исходного кода тестируемого приложения, написание и обработку модульных тестов.

Сборка и выполнение модульных тестов на целевом отладочном оборудовании значительно экономит силы и время разработчиков, а также повышает эффективность их работы.

Это особенно важно в условиях современного рынка встраиваемых приложений, когда сроки разработки неуклонно сокращаются. Значит, возрастает и вероятность ошибок. Как следствие, это способно привести к снижению прибыли и росту затрат на эксплуатационное обслуживание и исправление возникших проблем с использованием продукта.

Внедрение профессиональных инструментов автоматизации тестирования встраиваемых систем, таких как Atollic TrueVERIFIER, позволяет разработчикам автоматически генерировать наборы модульных тестов и выполнять их в наиболее правдоподобной среде эксплуатации — на целевой отладочной плате.

Автоматически генерируемые наборы модульных тестов, как правило, обрабатывают большое число потенциальных путей исполнения кода тестируемых функций. И эти тесты можно без каких-либо дополнительных усилий постоянно синхронизировать с последними изменениями кода.

В результате качество разрабатываемого ПО повышается, а потраченное время и объем работ сокращаются. Таким образом, специалисты получают в свое распоряжение мощный и эффективный инструмент тестирования.

С точки же зрения готового продукта это прекрасная возможность поставки на рынок в самые короткие сроки. Причем риск возникновения проблем у потребителей на этапе эксплуатации значительно снижается, затраты на техобслуживание сокращаются. Бюджет экономится, прибыль увеличивается.

Что касается конечного потребителя, он получает высококачественное ПО, которое может уверенно применять для решения своих самых важных и критичных задач.

Литература

1. Сергеева А. Возможности статического анализатора Atollic TrueINSPECTOR для повышения качества встраиваемых приложений // Компоненты и технологии. 2015. № 4.
2. Сергеева А. Тестирование работоспособности промышленного компьютера // Компоненты и технологии. 2014. № 2.
3. Об инструменте Atollic TrueVERIFIER. **www.atollic.com/trueverifier**
4. О среде разработки Atollic TrueSTUDIO. **www.atollic.com/truestudio**
5. Сергеева А. Применение реинжиниринга при проектировании встраиваемых систем. Часть 1 // Компоненты и технологии. 2014. № 9.
6. Сергеева А. Применение реинжиниринга при проектировании встраиваемых систем. Часть 2 // Компоненты и технологии. 2014. № 10.