ТЕСТИРОВАНИЕ ТРАНСЛЯЦИИ ФОРМАЛЬНЫХ МОДЕЛЕЙ

А. Ю. Подъячев[1], С. В. Афанасьев[2]

Санкт-Петербургский институт информатики и автоматизации РАН

СПИИРАН, 14-я линия ВО, д. 39, Санкт-Петербург, 199178

[1]<gedo@yandex.ru>,[2]<serg@sqlab.spb.su>

УДК 004.05

Подъячев А.Ю., Афанасьев С. В. **Тестирование трансляции формальных моделей** // Труды СПИИРАН. Вып. 3, т. 2. — СПб.: Наука, 2006.

Аннотация. *Формальное моделирование является важной частью процесса создания программного продукта на всех этапах разработки. Функциональное тестирование, в свою очередь, неотъемлемая часть обеспечения качества разработки программного продукта. Построение механизма трансляции формальных моделей между нотациями привело к необходимости контроля правильности передачи функционала между моделями. В статье описана постановка задачи для решения такого рода проблем на основе моделирования тестовых сценариев при помощи формальных моделей.* — Библ. 10 назв.

UDC 004.05

Podyachev A.Y., Afanasyev S.V. **Formal model translation testing** // SPIIRAS Proceedings. Issue 3, vol. 2. — SPb.: Nauka, 2006.

Abstract. *Formal modeling is very important part of program creation process in the all development steps. Functional testing, in one's turn, is integral part of quality assurance. Formal models translation mechanisms creating bring to necessity of validation control for functional transmission across models. This article describes methods for solving the problem by test scenarios modeling with formal models help.* — Bibl. 10 items.

1. Введение

В настоящее время в качестве средства построения тестовых сценариев параллельно написанию программного кода широко используется оболочка Eclipse [1]. Продукт Eclipse является проектом открытых ресурсов “Eclipse org.”, контролируемым комитетом по управлению данным проектом, который возглавляет компания IBM. Основное внимание уделяется параллельному метамоделированию. Под этим понятием следует понимать параллельное развитие метамодели проектирования и метамодели тестирования, которые в процессе дальнейшей декомпозиции приводят к соответствующим функциональным моделям проектирования и тестирования. Для определения и формализованного выражения различных видов тестирования используется нотация TTCN, которая интегрирована с Eclipse.

Нотация тестирования и контроля тестирования TTCN (Testing and Test Control Notation) [2] является языком, используемым для написания детализированных спецификаций тестов. TTCN использовался для написания тестовых спецификаций для многих видов приложений, включая мобильную связь (GSM, 3G, TETRA), беспроводные сети (Hiperlan/2), беспроводную телефонию DECT, широкополосные технологии (B-ISDN, ATM), платформы, основанные на CORBA и Интернет протоколы, такие как IPv6, SIGTRAN, SIP и OSP.

Последняя версия языка, TTCN версии 3 (TTCN-3), стандартизирована ETSI (ES 201 873 серии) и ITU-T (Z.140 серии)

Плюсами использования TTCN-3 является

- язык специально спроектирован для тестирования;

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- синтаксис и семантика операторов тестов TTCN-3 общедоступна и не привязана к особым языкам программирования;
- тесты TTCN-3 направлены на результаты тестирования и абстрагируются от особых деталей тестируемой системы;
- язык постоянно дорабатывается и совершенствуется. В образовательных и тренировочных целях он может быть рационализован и сокращен;
- делает возможным применение большинства методологий и стилей как корпоративного уровня, так и не стандартизованных.

Для оптимальной связи построения программной модели и модели тестирования рекомендуется использовать в качестве языка моделирования нотацию UML [3].

Диаграммы структурного системного анализа IDEF [4] также продолжают использоваться целым рядом организаций для построения и детального анализа функциональных моделей, существующих на предприятии бизнес-процессов, а также для разработки новых бизнес-процессов. Также некоторые нотации, в частности нотация IDEF-0, используется для моделирования программного обеспечения. Основной недостаток данной нотации связан с отсутствием явных средств для объектно-ориентированного представления моделей сложных систем. Ограниченные возможности нотации IDEF-0, применительно к реализации соответствующих графических моделей объектно-ориентированных программ, существенно сужают диапазон решаемых с ее помощью задач. Именно отсутствие таких ограничений, а также наличие дополнительных возможностей в UML, в виде генерации программного кода по графической модели алгоритма, позволяет говорить о необходимости трансляции формальных моделей. В данном случае речь идет о трансляции моделей на основе IDEF-0 в формальные модели на основе UML. Очевидно, что в процессе разработки такого программного решения немаловажную роль играет контроль правильности передачи функционала модели. Для удовлетворения требований контроля качества трансляции будут использоваться методы и средства функционального тестирования.

2. Функциональное тестирование

Функциональное тестирование (functional testing) — процесс верификации соответствия функционирования продукта его начальным спецификациям [5]. Характерным примером может быть проверка того, что программа подсчета выплат по банковской ссуде выдает корректные выкладки на любые введенные сумму ссуды и срок ее возврата. Обычно подобные проверки проводятся вручную, иногда к этому подключаются конечные пользователи в качестве бета-тестеров. Однако, по мере сложности программных систем, комбинации различных входных параметров и поддерживаемых операционных систем нередко исчисляются десятками и сотнями. В такой ситуации функциональное тестирование требует значительного объема ресурсов; естественным выходом является автоматизация этого процесса [6,7]. В связи с тем, что речь пойдет о трансляции формальных моделей, наиболее подходящим можно считать использование тестирования на основе этих моделей.

Преимущества тестирования на основе моделей заключаются в том, что

- тесты на основе спецификации функциональных требований более эффективны, так как они в большей степени нацелены на проверку функциональности, чем тесты, построенные только на знании реализации;

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

- на основе формальных спецификаций можно создавать самопроверяющие (self-checking) тесты, так как из формальных спецификаций часто можно извлечь критерии проверки результатов целевой системы.

Модели обычно проще реализации, можно предположить, что тесты, хорошо «покрывающие» модель, слишком бедны для покрытия реальных систем. Поэтому требуется некоторое количество широких экспериментов в реальных проектах для определения качества тестового покрытия.

Модель — некоторое отражение структуры и поведения системы [9]. Модель может описываться в терминах состояния системы, входных воздействий на нее, конечных состояний, потоков данных и потоков управления, возвращаемых системой результатов и т.д. Для отражения разных аспектов системы применяются и различные наборы терминов. Формальная спецификация представляет собой законченное описание модели системы и требований к ее поведению в терминах того или иного формального метода. Для описания характеристик системы можно воспользоваться несколькими моделями в рамках нескольких формализмов, например разных типов UML диаграмм. Обычно, чем более общей является нотация моделирования, тем больше трудностей возникает при тестировании программы на основе модели/спецификации, описанной в этой нотации. Одни нотации и языки больше ориентированы на доступность и прозрачность описания, другие — на последующий анализ и трансляцию, в частности, трансляцию спецификации в тестовый сценарий. При помощи нотации TTCN-3 формальные описания построенных тестовых сценариев будут стандартизированы для дальнейшего использования.

Суть предлагаемого подхода заключается в следующем решении: любая формальная модель в нашем контексте подлежащая трансляции отображает не что иное, как функциональный алгоритм (диаграммы действия или состояния). Адаптировав к данной ситуации методы построения тестового сценария по формальной модели, мы получим алгоритм проверки для конкретных функциональных моделей. При параллельной проверке функционала исходной модели перед трансляцией и полученной модели после трансляции мы, по отмеченным заранее, так называемым *критическим точкам*, можем сопоставить результаты проверки на различных этапах тестирования. В случае совпадения результатов проверки функционала в ходе двух параллельных проверок на всех этапах передачу функциональности формальной модели можно признать корректной. В случае возникновения разночтений результата можно локализовать сегмент некорректно переданного функционала. В данном случае степень детализации, т.е. частота расстановки критических точек является одним из важных критериев, влияющим на общее качество контроля и степень простоты локализации проблемной области.

На основании проведенной серии опытов по проверке оттранслированных моделей, можно будет принять решение о корректности работы транслятора в целом. Так же можно будет систематизировать погрешность трансляции, и в случае когда программно улучшить передачу функционала не будет представляться возможным, эта величина послужит значением, которое придется принимать во внимание и конечному пользователю.

Для полноты определения качества передачи функциональности необходимо учесть ряд немаловажных метрик, от которых напрямую зависит качество программных продуктов в целом [10]. Такими метриками являются: циклическая сложность, размер класса, отклик для класса, недостаток единства, связь меж-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

ду объектами классов, глубина дерева наследований, число потомков. Далее приведенные метрики будут описаны подробнее.

3. Используемые метрики оценки качества.

Циклическая сложность используется для того, чтобы оценить сложность алгоритма в методе. Это число испытательных случаев, которое необходимо для всесторонней проверки этого метода. Формула для вычисления циклической сложности — это число рёбер минус число узлов плюс 2. Для последовательности, где имеется только один путь, необходим только один испытательный случай. Условие if, однако, имеет два варианта; если условие истинно, проверяется один путь, если ложно — второй.

Метод с низкой циклической сложностью лучше. Это может означать уменьшение тестирования и увеличение простоты кода. Циклическая сложность не может использоваться для измерения сложности класса из-за наследования, но циклические сложности отдельных методов могут быть объединены с другими мерами для оценки сложности класса. Хотя этот показатель определённо применим к сложности, он также связан со всеми другими атрибутами.

Размер класса используется, чтобы оценить простоту понимания кода разработчиками. Размер может быть измерен разными путями. Среди них подсчёт всех строк программы, числа инструкций, числа пустых строк и числа строк комментария. Строки программы (LOC) считает все строки. Не пробел, не комментарий (NCNB) иногда называется исходными строками кода и считает все строки, которые не являются комментариями и пробелами.

Пороги для оценивания значения **размера** изменяются в зависимости от используемого языка программирования и сложности метода. Однако, так как размер влияет на простоту понимания разработчиками, классы и методы большого размера всегда связаны с высоким риском.

Отклик для класса считает набор всех методов, которые могут быть вызваны в ответ на сообщение, обращённое к объекту класса или посланное одним из методов класса. Он включает все методы, доступные в иерархии классов. Эта метрика связана со сложностью класса через число методов и количеством связей с другими классами. Чем больше число методов, которые могут быть вызваны классом через сообщение, тем больше сложность класса. Если в ответ на сообщение может быть вызвано большое количество методов, тестирование и отладка класса усложняются, так как это требует большего уровня понимания от тестировщика. Худший случай значения возможных откликов помогает распределить время тестирования.

Недостаток единства измеряет несходство методов в классе по эталонной переменной или атрибутам. Высоко единый модуль должен быть автономным; высокое единство указывает на хорошее подразделение класса. Недостаток единства или низкое единство увеличивает сложность и вероятность возникновения ошибок в течение процесса разработки. Классы с низким единством, вероятно, могут быть подразделены в два или более подкласса с большим единством.

Связь между объектами классов определяет количество других классов, с которыми связан этот класс. Это подсчитывается определением числа удалённых не наследованных связанных классовых иерархий, от которых класс зависит. Чрезмерная связность вредна для модульного проекта и не позволяет его многократное использование. Чем больше число связей, тем выше чувстви-

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

тельность к изменениям в других частях проекта, и поэтому обслуживание усложняется.

Глубина класса в иерархии наследований равна максимальному числу шагов от класса до корня дерева и измеряется числом классов-предков. Чем глубже класс в иерархии, тем большее число методов, возможно, он унаследует, и тем сложнее предсказать его поведение. Более глубокие деревья делают проект более сложным, так как используется большее количество методов и классов, но одновременно возрастает возможность повторного использования унаследованных методов. Для вычисления DIT используется число унаследованных методов (NMI).

Число потомков — это число непосредственных подклассов, принадлежащих классу в иерархии. Это индикатор влияния, которое класс может иметь в проекте и системе. Чем больше число потомков, тем больше вероятность неподходящей абстракции родителя и, вероятно, имеет место случай неверного употребления подклассификации. Но чем больше количество дочерних классов, тем больше вероятность многократного использования, так как наследование — форма многократного использования. Если класс имеет большое число потомков, это может потребовать большого количества тестов для методов этого класса, что отражается на времени тестирования.

4. Заключение

Разработка методов тестирования предполагает исследование и использование результатов достигнутых в области функционального тестирования программных систем. Будут рассмотрены основные стандарты оценки качества программ и проанализированы существующие метрики качества. Также будут использоваться методы и свойства формальных моделей применяемых для моделирования объектно-ориентированного программного обеспечения.

Обязательным условием решения основной задачи является изучение реализованного в программном комплексе метода трансляции между моделями, что позволит рассмотреть вопрос с точки зрения понимания внутренних процессов трансляции. Для выявления необходимых пост- и пред- условий проверки транслятора, в процессе работы будут исследованы методы автоматической генерации тестовых сценариев на основе формальных моделей.

Как основой подход к решению задачи можно определить систематизацию тестовых критериев, выявление общих закономерностей проверки трансляции и выражение их в виде формальной структуры.

Результат этой работы поможет создать предпосылки для дальнейшего развития методов анализа моделирования и разработки новых критериев оценки качества программных систем. Значением результатов будет являться возможность их применения не только в области разработки программного обеспечения, но и в других областях знаний, таких как: планирование бизнес процессов, моделирование производственных процессов и других областях, использующих формальные модели.

Литература

1. IDE Eclipse [Электронный ресурс] // <http://www.ubs.ru/ws/eclipse.html> (по состоянию на 20.03.2006).
2. OpenTTCN [Электронный ресурс] // <http://www.openttcn.com> (по состоянию на 20.03.2006).

Труды СПИИРАН. 2006. Вып. 3. Т. 2. ISSN 2078-9181 (печ.), ISSN 2078-9599 (онлайн)
SPIIRAS Proceedings. 2006. Issue 3. V. 2. ISSN 2078-9181 (print), ISSN 2078-9599 (online)
www.proceedings.spiiras.nw.ru

3. UML. [Электронный ресурс] // <http://www.omg.org/technology/documents/formal/uml.htm> (по состоянию на 20.03.2006).
4. Технологии системного моделирования. [Электронный ресурс] // <www.idefinfo.ru/> (по состоянию на 20.03.2006).
5. *Канер С., Фолк Д., Нгуен Е.К.* Тестирование программного обеспечения. Киев: ДиаСофт, 2000. 544 с.
6. *Дастин Э., Рэшка Д., Пол Д.* Автоматизированное тестирование программного обеспечения. Москва: Лори, 2003. 592 с.
7. *Robinson H.* Intelligent Test Automation, Software Testing and Quality Engineering, Issue Sep/Oct 2000, SW Quality Engineering. [Электронный ресурс] // <www.stqemagazine.com> (по состоянию на 20.03.2006).
8. *Kruchten P.* The Rational Unified Process. Rational Suite documentation. An introduction. [Электронный ресурс] // <http://www.interface.ru/rational/rup01_t.htm> (по состоянию на 20.03.2006).
9. *Петренко А., Бритвина Е., Грошев С., Монахов А, Петренко О.* Тестирование на основе моделей. [Электронный ресурс] // <http://www.citforum.ru/SE/testing/model/> (по состоянию на 20.03.2006).
10. *ISO/IEC*, TR 9126-3, 2003 Software engineering - Product quality - Part 3 Internal metrics.