УДК 004

# СОСТАВЛЕНИЕ ТЕСТОВЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ “TEST CASE” НА ОСНОВЕ ОБЪЕКТНО-ОРИЕНТИРОВАННОЙ МОДЕЛИ

***Зиновьев Д.А., Фонотов А.М.***
*Донецкий Национальный Технический Университет*
*Кафедра автоматизированных систем управления*
*e-mail:da451@mail.ru*

***Аннотация***

***Зиновьев Д.А., Фонотов А.М. Составление тестовых последовательностей «Test Case» на основе объектно-ориентированной модели.*** *В статье рассматривается вопрос, и методы составления тестовых последовательностей, которые используют объектно-ориентированную модель системы. Проведен анализ ранее использованных методик, проанализированы их достоинства и недостатки, рассмотрены пути автоматизации процесса оптимальных тестовых последовательностей на основе объектно-ориентированных моделей.*

***Общая постановка задачи.*** Процесс разработки программного обеспечения состоит из 5 этапов: выдвижение требований, анализ, проектирование, разработка, тестирование [1].

Производство качественного программного обеспечения является основной задачей разработчиков. Но помимо самой разработки, много времени и средств уходит на проверку выпускаемого продукта. Процесс тестирования программного обеспечения позволяет дать оценку качеству программного продукта, оценить готовность его к внедрению. На сегодняшний день методы тестирования не позволяют выявить все возможные «неисправности» при рассмотрении крупных информационных систем и дать однозначный ответ о корректности работы программного обеспечения [1]. Но в основном от качества проверки будет, зависит, насколько хорошим и популярным будет данный продукт.

Тестируя довольно не большую модель, можно проверить практически все варианты выполнения системы. Но если система большая, то количество тестов и время их выполнения может стать серьезной проблемой для разработчиков и тестировщиков. В связи с этим появляется задача оптимизации тестовых наборов. Для этой цели используются критерии тестового покрытия. Критерии тестового покрытия это определенный набор условий, выполнив который можно сказать, что система успешно прошла тестирование. На данный момент считается что хороший процент покрытия 60-70% [2].

Перечислим самые распространенные критерии тестового покрытия:

- Statement coverage – в тестовом наборе T, каждое состояние будет выполнено хотя бы раз.
- Branch coverage – в тестовом наборе T, каждый оператор ветвления должен быть выполнен как для значения true так и для значения false.
- Conditional coverage – каждое условие в графе должно быть выполнено как для true так и для false
- Path coverage – в тестовом наборе T, каждый переход должен быть пройден хотя бы раз. [2]

На сегодняшний день вопросу тестирования, а именно его автоматизации, уделяется большое внимание. Создаются специальные программы для тестирования продукции, которые используют код программы или модель системы. Но все, же основная часть продуктов рассчитаны на работу именно с программным кодом. В связи с тем, что существует огромное количество языков программирования и принципов построения кода

программ, тестирование на основе программного кода является сложно выполнимой задачей. Для того что бы сделать процесс тестирования универсальным, следует использовать для построения тестов объектно-ориентированную модель системы. Можно выделить 3 основные причины, почему следует пользоваться моделью проекта:

1) традиционное программное тестирование включает в себя только статистическое рассмотрение кода, которое не достаточно для тестирования динамического поведения системы;

2) использование кода для проверки системы является сложной и трудоемкой задачей;

3) модель помогает лучше понять систему и найти данные для тестирования после просмотра модели, чем та же работа с кодом.

Генерация тестовых последовательностей, основанная на объектно-ориентированных моделях, может быть начата на самых ранних этапах создания программного продукта и позволяет выполнить одновременно написание и проверку продукта.

Наиболее универсальным способом тестирования, является последний рассмотренный вариант – формирование тестов на основе объектно-ориентированных моделей. Для построения объектно-ориентированных моделей чаще всего используют язык UML. Диаграмма взаимодействия, диаграмма классов, диаграмма состояний позволяют рассмотреть динамику взаимодействия объектов в системе и возникающие при этом ветвления и альтернативы потока событий.

Различают 3 вида тестирования :

1) тестирование черного ящика;
2) тестирование белого ящика;
3) тестирование серого ящика;

В основе тестирования объектно-ориентированных моделей лежит так называемые серый ящик, это вид тестирования, когда разработчику тестов доступен не весь код, но алгоритм процесса и все возможные условия которые могут повлиять на конечный результат выполнения программы.

При рассмотрении вопросов тестирования на основании объектно-ориентированных моделей Test Case представляет собой набор констант, значений переменных, которые приводят систему в одно из состояний. Для того что бы понять работает ли система правильно, сравнивается test oracle – ожидаемый результат системы после тестирования и actual result – результат который тестировщик получает после тестирования. На основании анализа полученных результатов делают выводы был ли тест пройден.

***Представление моделей.*** Любую модель системы или программы можно рассматривать как функцию P. P: S → R, где S это набор возможных входных значений и R это набор возможных результатов. Более формально S это набор векторов $x= (d_1,d_2…d_n)$, так чтобы $d_i \in D_{xi}$ где $D_{xi}$ является областью входных переменных $x_i$.

Поток управления графа (Control Flowgraph) модели P это направленный граф G=(N,E,s,e) который состоит из набора узлов N и набора ребер $E=\{(n,m)|n,m \in N\}$ соединенных с узлами. В каждом графе есть два специальных узла входной узел s и выходной узел e [3].

Каждый узел можно считать базовым блоком, который интерпретируется как последовательность инструкций. Выполнение блока можно понимать как выполнения последовательности инструкций в этом блоке. Ребро между двумя узлами n и m, соответствует возможному переходу из n в m. Каждое ребро представляет собой предикат с условием. Предикат может быть и без условия – пустой предикат, значение которого всегда true.

Путь $p=(p_1,p_2…p_{qp})$ - это последовательность узлов, где $p_{qp}$ это последний узел пути. Выполнение модели P с входными параметрами $x_i$ означает, что сценарий с входными

параметрами $x_i$ приводит к прохождению по пути p. Путь называют осуществимым, если существует $x_i \in S$ пересекающий путь, иначе его называют не осуществимым.

На рисунке 1 представлена схема генерации тестовых данных предложенная Бейзером.


Рисунок 1 - Схема генерации тестовых данных.

Исходя из утверждений Бейзера, были созданы описания для диаграмм последовательности, представленные ниже[2].

Каждая диаграмма представляет собой запись следующего вида $D=(A,T,F,C,a_1,a_f)$, где:

- $A=\{a_1,a_2 \ldots a_m\}$ это набор состояние диаграммы деятельности
- $T=\{t_1,t_2 \ldots t_n\}$ это набор переходов диаграммы деятельности
- $C==\{c_1,c_2 \ldots c_n\}$ это набор граничных условий, $c_i$ соответствует $t_i$. Con это отображение значения $t_i$, то есть $Con(t_i)= c_i$.
- $F \subseteq \{A \times C \times T\} \cup \{T \times C \times A\}$ это поток связей между действиями и переходами
- $a_1 \in A$ начальный узел, $a_f \in A$ конечный узел

Для генерации Test Cases нам необходимо сравнить выполнение программы с динамическим выполнением диаграммы деятельности[2].

***Методика формирования Test Case.*** Существует множество различных алгоритмов для построения тестов по модели системы. И зачастую данная процедура содержит три основных шага:

1) формирование системы и ввод ограничений и условий;
2) интерпретация модели в виде графа;
3) составление тестовых последовательностей(test case);

Рассмотрим эти шаги подробнее.

*Формирование системы и ввод ограничений и условий*. Для того что бы начать процесс составления Test Cases необходимо сперва составить модель проекта. Для разработки проекта будем использовать UML. Появление OCL (Object Constraint Language) дало возможность накладывать дополнительные условия в UML-модели. Условия, задаваемые с помощью OCL, позволяют рассмотреть диаграммы последовательности и деятельности, как расширенные сценарии поведения системы.

*Интерпретация модели в виде графа*. Часто для формирования тестовых последовательностей используют Структурированный составной граф (Structured Composite Graph). Это делается для того что бы систематически исследовать потоки управления модели. Информация, которая храниться в диаграмме преобразуется, и храниться в структурированном составном графе. Поочередно все состояния диаграммы исследуются и располагаются на графе. Для автоматизации процесса формирование тестовых

последовательностей представление диаграммы в виде графа является очень удобным. Рассматриваемая модель должна содержать информацию и требования к возможным значениям переменных и к входным данным, которые приводят к изменению состояния системы.

*Составление тестовых последовательностей(Test Case)*. Задача состоит в том, чтобы сгенерировать такой набор входных данных $x_i$ для модели P и пути u, что бы набор $x_i$ привел к движению по сценарию u. Выделяют три класса методов генерации тестовых последовательностей: генерация случайных тестовых данных (random), целенаправленная генерация данных (goal-oriented), генерация данных с использованием пути (path-oriented).

*Генерация случайных тестовых данных (random)*. Данный метод является самым простым для составления тестовых данных. Он может быть применен для любого типа данных. Но данный метод имеет маленькую вероятность нахождения ошибок, а так, же маленькое тестовое покрытие. Данные, которые сгенерированы, могут не удовлетворять определенным условиям модели.

*Целенаправленная генерация данных (goal-oriented)*. Этот метод намного эффективнее случайной генерации. В место того чтобы сгенерировать данные которые позволяют рассмотреть модель от начала до конца, метод генерирует такой набор данных который позволит пересечь заданный неспецифический(unspecific) путь.

Одним из методов, использующих эту методику, является цепочный подход. Метод цепочки ищет значения, для предикатов ветви исходя из накладываемых условий. При этом решается задача идентифицировать цепочку узлов, которые необходимо выполнить, что бы дойти до выбранного узла. Эта цепочка строиться итеративно в процессе выполнения.

*Генерация данных с использованием пути (path-oriented)*. Данный метод является самым результативным из перечисленных. Он не предоставляет возможность выбора пути из определенного множества возможных путей, однако дает возможность выбрать один определенный путь. В этом этот метод чем-то похож на целенаправленную генерацию. Использование выделенных путей приводит к большему покрытию модели. С другой стороны сложнее найти тестовые данные.

На рисунке 1 изображена схема генерации данных с использованием пути. Важным компонентом схемы является именно выбор пути (Path selector). Выбирается такой набор путей, который будет удовлетворять выбранным критериям покрытия.

На рисунке 2 представлена полная схема генерации Test Cases для диаграммы деятельности.

**Пример**. Рассмотрим пример составления Test Case. Рассмотрим одну из разновидностей Path coverage criterion. Данный критерий называется Activity Path Coverage Criterion.

Для нахождения всех возможных путей используется алгоритм прохождения графа depth-first search и breadth-first search. Необходимо проходить пути начиная с начального узла и до конечного узла. Для данного графа существует 3 основных пути проходящих от начального узла и до конечного.

$AP_1$=1→2→3→4→5→6→2→3→4→5→6→7

$AP_2$=1→2→3→4→5→6→2→3→8→9→10→7

$AP_3$=1→2→3→4→5→6→2→3→8→9→11→12→13→14→15→16→17→7

Допустим $AP_i$ это один из путей в наборе возможных путей графа. Разделим на набор под путей $AP_i$=$P_1$ $P_2$ $P_i$ $P_m$ $P_i$ $P_n$ где $P_1$ $P_2$ $P_i$ $P_m$ $P_i$ $P_n$ это набор под путей $AP_i$ и Pi выполнимы.

$P_d(AP_1)$= {1 2 3 4 5 6 7}

$P_d(AP_2)$= {1 2 3 8 9 10 7}

$P_d(AP_3)$= {1 2 3 8 9 11 12 13 14 15 16 17 7}

После того как были найдены все возможные пути, надо найти значения переменных позволяющие пересечь эти пути.

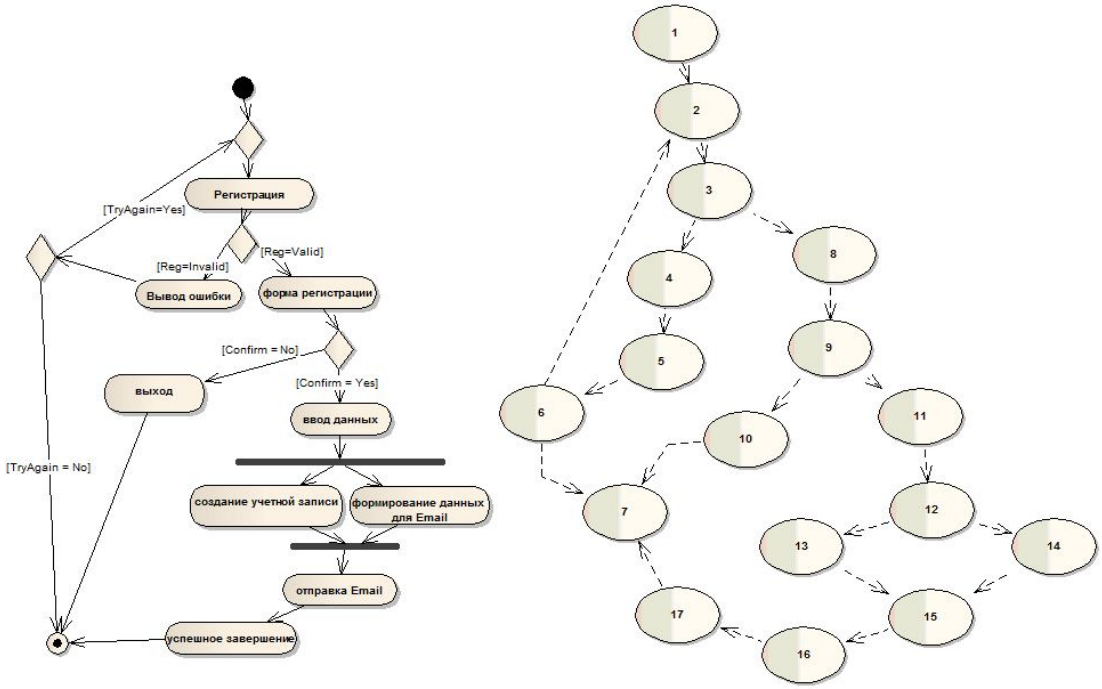


Рисунок 2 – диаграмма деятельности «Регистрации» и ее граф

Таблица 1. Test Cases для примера

| Test Case № | Значения переменных | Последовательность действий |
|---|---|---|
| 1 | Reg=invalid, TryAgain=Yes, Reg=invalid, TryAgain=No | регистрация, вывод ошибки, регист., вывод ошибки |
| 2 | Reg=invalid, TryAgain=Yes, Reg=valid, Confirm=No | регистрация, вывод ошибки, регист., выход |
| 3 | Reg=invalid, TryAgain=Yes, Reg=valid, Confirm=No | Регист., вывод ошибки, регист., форма регистр., выход |
| 4 | Reg=invalid, TryAgain=Yes, Reg=valid, Confirm=Yes | Регист., вывод ошибки, регист., форма регистр., ввод данных, создание уч. записи, форм. Email, отправка Email, успешное завершение |

**Выводы.** Составление Test Cases для объектно-ориентированных моделей является много обещающим направлением исследований, ведь это позволит начать процесс тестирования с самого начала разработки и реализации проекта.

Используя методы тестирования, основанные на составлении пути и набор критериев покрытия, выбрать набор тестов, который будет удовлетворять условиям тестирования, и удовлетворять заданным требования работы системы.

Создание подобной системы, которая сможет составить полный набор тестов для ПО имея в наличии модель, является универсальной и наглядно показывает все возможные случаи событий.

## Список литературы

1. Journal of Object Technology 2008// A Novel Approach to Generate Test Cases from UML Activity Diagrams.

2. Journal of Object Technology 2008// Automatic Test Data Synthesis using UML Sequence Diagrams

3. A survey on automatic test data generation. In Proceedings of the Second Conference on Computer Science and Engineering in Linkoping, pages 21-28.ECSEL, October 1999.