Модел. и анализ информ. систем. Т. **21**, № **6** (2014) 144–154

УДК 004.415

Инкрементальный подход к технологии создания тестов для индустриальных проектов

Дробинцев П.Д.[1], Котляров В.П.[1], Никифоров И.В.[1], Летичевский А.А.[2]

[1] *Санкт-Петербургский государственный политехнический университет, 195251, Россия, г. Санкт-Петербург, ул. Политехническая, 29*
[2] *Институт кибернетики им. В.М. Глушкова НАН Украины, 03680 МСП, Украина, г. Киев, просп. Академика Глушкова, 40*

e-mail: [1]*vpk@spbstu.ru,* [2]*let@cyfra.net*

получена 30 сентября 2014

Ключевые слова: тестирование на основе моделей, верификация, автоматизация тестирования, сокращение пространства состояний

Статья посвящена описанию технологии, позволяющей сократить трудозатраты на создание тестов для промышленных программных проектов, за счет использования инкрементального подхода. Основная проблема, решенная в данной работе, связана с полной автоматизацией фазы дизайна тестовых сценариев и сокращением количества тестов, необходимых для обеспечения качества программного продукта. Предлагаемая в работе технология позволяет решить указанные проблемы за счет совместной работы дизайнера тестовых наборов и заказчика с использованием формальных моделей, методов символьной верификации и автоматической генерации тестовых наборов на базе использования инструментария VRS/TAT.

1. Введение

В настоящее время в связи с ростом мощности аппаратных и программных средств в списке дефицитных ресурсов разработки программного продукта (ПП) первые места заняли трудоемкость, надежность и эффективность. Трудоемкость определяет необходимый объем инвестиций и сроки производства ПП, надежность – защищенность ПП от сбоев и дефектов, а эффективность – пропускную способность обработки информации.

В работе рассмотрен подход к уменьшению трудоемкости разработки промышленного ПП, основанный на применении инкрементальной технологии. Основная задача технологии – создание надежного и корректного относительно требований ПП с наименьшими ручными усилиями. Особые проблемы создает промышленный ПП, отличающийся большими размерами и особыми требованиями к надежности и организации производства.

Основной задачей создания эффективной промышленной технологии является создание инструментария, обеспечивающего полную автоматизацию производства ПП, и технологии его применения. В настоящей работе описывается часть решения общей задачи, заключающаяся в автоматизации этапа проектирования, на котором после верификации и тестирования поведенческой модели ПП по модели генерируется целевой код тестового набора, обеспечивающего 100% контроль функциональности приложения в соответствии с различными критериями.

Поведенческая модель, используемая в проектировании ПП, – это спецификация на расширенном языке UCM [1]. Детальное сравнение нотаций приведено в [20]. Размер модели определяется числом элементов в множестве UCM диаграмм, специфицирующем поведенческую модель. Модель ПП среднего размера содержит от 50 до 500 UCM элементов, модель большого размера – от 500 до 10000 UCM элементов.

Основной особенностью производства промышленного ПП является практическая необходимость решать производственные задачи создания ПП по частям. Это означает применение инкрементального подхода на этапе проектирования, который требует декомпозиции ПП на компоненты, начиная с ранних фаз проектирования, покомпонентного анализа поведения и обработки отдельных компонент, а также композиции результатов анализа и распространения их на весь ПП.

Подход покомпонентного проектирования известен в технологии разработки ПП уже достаточно давно [2, 3], причем в настоящее время он активно развивается в рамках MDE (Model Driven Engineering). Основной упор в упомянутых работах сделан на разработке языков [5, 7] и способов спецификации модулей, межмодульных интерфейсов и протоколов [5], обеспечивающих удобство композиции и декомпозиции на модули [4, 6], обоснование корректности проектируемых моделей и введения ограничений на используемые формализмы и процедуры проектирования [8]. Тем не менее, в существующем подходе присутствует ряд неразрешенных проблем:

1. Проблема сохранения в спроектированной модели семантики исходных требований.

2. Проблема интеграции в процессе обоснования спроектированной модели доказательных и экспериментальных подходов т.е. верификации и тестирования.

3. Проблема полной автоматизации в единой среде всех процессов от формализации требований и проектирования моделей до генерации исполняемого кода тестов и автоматического тестирования.

4. Проблема применения разработанных подходов к промышленным ПП большого размера.

Особенностью предлагаемого подхода является решение перечисленных проблем на этапе формализации требований за счет совместной работы проектировщика и заказчика на основе использования прозрачной для обеих сторон нотации UCM [1]; за счет интеграции символьной верификации с автоматической генерацией тестовых наборов [9]; за счет использования эффективной технологии и интегрированного инструментария VRS/TAT [10]; за счет опыта применения описываемого подхода к промышленным ПП из области телекоммуникаций.

2. Формальное определение поведений программного продукта

Спецификация приложения на расширенном языке UCM задается графом из UCM элементов, а модель поведения множеством сценариев на этом графе. Если каждому UCM элементу сопоставить тройку Хоара [11] или помеченный базовый протокол Летичевского [12], то поведенческая модель может быть представлена в виде некоторой транзиционной системы S. Поведение системы S в состоянии α обозначается как S_α. А.А. Летичевским доказано, что уравнение для S_α^∞ имеет следующий вид:

$$S_\alpha^\infty = \sum_{p \in P(\alpha)} proc(p) * (T(\alpha, p) : \Delta) * S_{T(\alpha,p)}^\infty. \tag{1}$$

В этой формуле $P(\alpha) = \{p \in P_{inst} | \alpha \to pre(p)\}$, P_{inst} – множество конкретизированных протоколов, $proc(p)$ – процесс базового протокола, $T(\alpha, p) = Tr(\alpha, post(p))$, $Tr(\alpha, \beta)$ – предикатный трансформер [17]. На его вход подаются две формулы α и β, а на выходе порождается новая формула γ, которая должна быть такой, что $\gamma \to \beta$. Данное условие необходимо для возможности применения последующих базовых протоколов, предусловие которых является неявным следствием постусловия β.

Следующее определение включает в себя конечные поведенческие сценарии (трассы) с успешным завершением. Выделим множество P^0 заключительных протоколов и положим $P^1 = \frac{P}{P^0}$. Предполагается, что заключительные протоколы могут завершать работу системы и не требуют ее обязательного продолжения. Уравнение для полной системы имеет вид:

$$S_\alpha = \sum_{p \in P^1(\alpha)} proc(p) * (T(\alpha, p) : \Delta) * S_{T(\alpha,p)} + \sum_{p \in P^0(\alpha)} proc(p) * (T(\alpha, p) : \Delta) * (S_{T(\alpha,p)} + \Delta). \tag{2}$$

Подмножество множества конечных трасс, удовлетворяющее некоторому критерию покрытия [13], представляет собой набор сценариев, пригодный для генерации тестового набора, удовлетворяющего выбранному критерию.

3. Последовательность UCM-элементов – поведенческий сценарий

Если зафиксировать некоторое начальное состояние системы S_0, которое включает состояние среды и состояния всех погруженных в нее агентов [14], тогда можно получить все возможные трассы (сценарии функционирования системы) как последовательности вида:

$$S_0 \xrightarrow{B_1(n_1,m_1)} S_1 \xrightarrow{B_2(n_2,m_2)} ...S_n, \tag{3}$$

где $B_1(n_1, m_1), B_2(n_2, m_2), \ldots$ – базовые протоколы, кодирующие соответствующие UCM элементы. При этом $n_1, n_2, \ldots$ – имена ключевых агентов (ключевой – фикси-

рованный агент базового протокола, состояние которого в данном базовом протоколе разрешено изменять), $m_1, m_2, \ldots$ – наборы значений остальных параметров, выполняющих предусловия базовых протоколов. Оценка количества возможных сценариев зависит от числа UCM элементов и от количества связей между элементами. Хотя значение оценки конечно, тем не менее, оно слишком велико для целей тестирования, в том числе для генерации тестового набора для ПП среднего и тем более большого размера.

4. Структурная модель программного продукта

UCM диаграммы, специфицирующие поведение модели ПП, представляют модель в виде системы вложенных компонент, описывающих солидарное поведение. Каждая компонента кодируется элементом Stub, являющимся ссылкой на вложенную диаграмму, специфицирующую поведение этой компоненты. В соответствии с общими законами композиции [15] каждая диаграмма строится так, чтобы предельная сложность поведения инкапсулировалась внутри диаграммы, а вне ее использовался интерфейс с небольшим количеством входов и выходов.

Структурная модель или дерево компонент (дерево Stub), образует структуру подчиненности диаграмм. Дерево Stub можно построить автоматически при обходе UCM диаграммы. На рис. 1 приведен пример построенного дерева компонент UCM диаграммы.

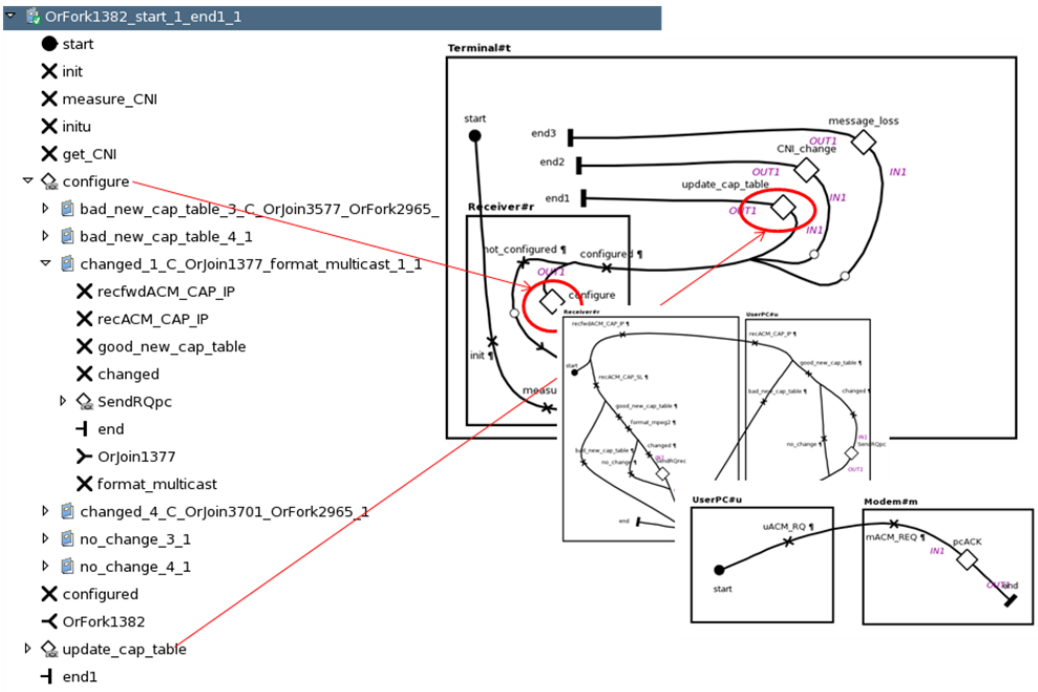

Рис. 1. UCM и ее дерево компонент

5. Анализ поведения компонент

Проектирование и анализ возможных трасс через обход дерева компонент является трудоемкой и сложной задачей для ПП среднего и большого размера. Намного проще реализовать подобный анализ покомпонентно. Действительно, если рассматривать диаграмму каждого Stub в отдельности, то для нее можно автоматически построить одноуровневое множество путей, покрывающих все ветви Stub при условии, что встречающиеся на путях вложенные Stub кодируются ссылками на соответствующие Stub-диаграммы.

В результате автоматически в границах Stub получаем множество путей, полностью покрывающих его поведение по критерию ветвей. Если сгенерированные для каждого Stub пути зафиксировать в отдельных директориях с соответствующими именами, а созданные директории разместить на позициях соответствующих Stub в дереве компонент, то дерево станет хранилищем всех возможных фрагментов поведенческих трасс приложения (рис. 2). Причем для получения конкретной трассы необходимо только выбрать и пометить путь в каждой из директорий и склеить их вместе.

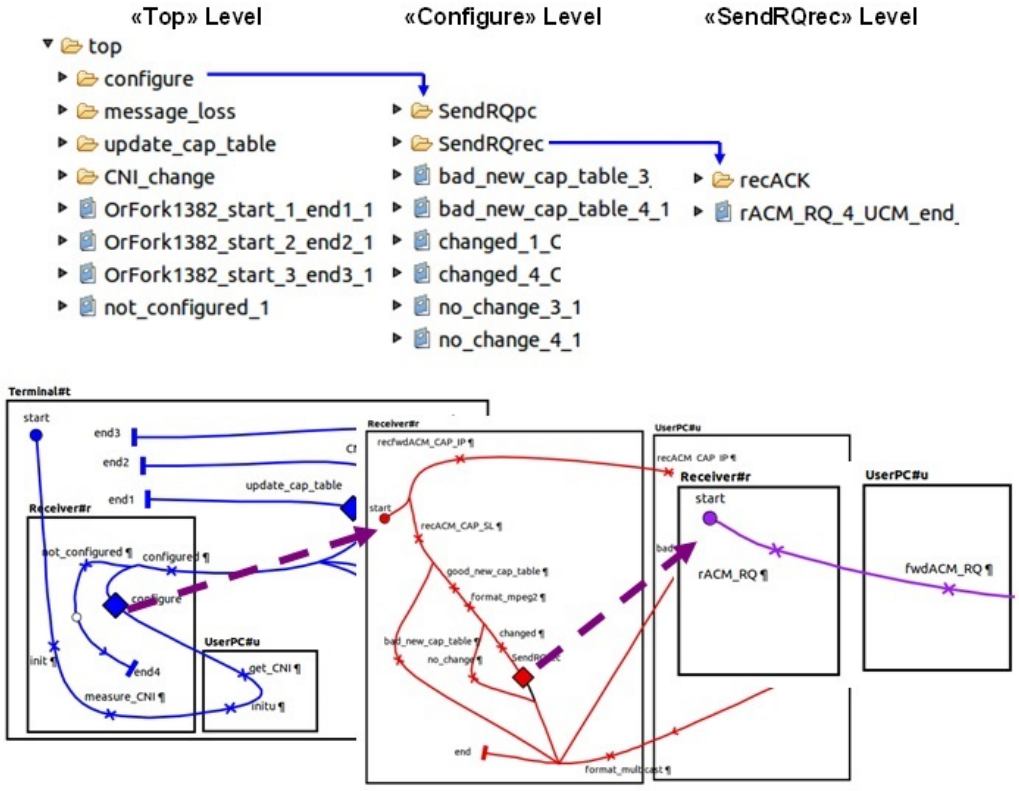

Рис. 2. Дерево как хранилище возможных путей поведения системы

6. Получение тестовых сценариев методом склейки

Тестовый сценарий при структурном представлении UCM диаграммы получается методом подстановки (вклейки) в пути верхних уровней, подчиненных путей из

тех Stub-директорий дерева компонент, на которые в путях верхних уровней есть ссылки. Если Stub в дереве компонент имеют по одному входу и одному выходу, то для получения полного тестового сценария достаточно выбрать конкретный путь в каждом Stub, на который есть ссылка в выбранном пути Stub верхнего уровня (рис. 3). Если в конкретном сценарии встречается несколько элементов $Stub_i (i \in [1..n])$ и каждый $Stub_i$ содержит n_i путей, то число возможных сценариев ограничено огромной величиной, связанной с перебором вариантов сценариев:

$$N = \Pi n_i (i \in [1..n]). \tag{4}$$

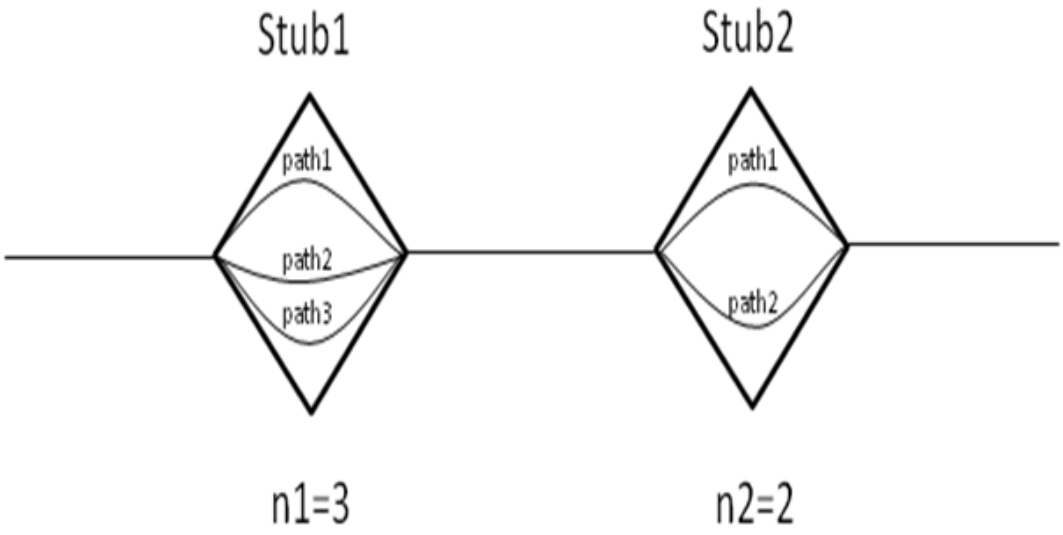

Рис. 3. Пример фрагмента UCM диаграммы с шестью возможными сценариями

7. Классификация путей по входам и выходам

Множество путей, характеризующих поведение Stub-диаграммы с несколькими входами или выходами, следует автоматически классифицировать по двум признакам: по принадлежности пути к некоторому StarPoint и EndPoint. Подобная классификация разделяет все множество поведений внутри Stub, уже покрытое путями, на интерфейсные классы, что при создании тестовых сценариев позволяет для вариантов путей прохода через Stub использовать пути, принадлежащие разным классам (рис. 4).

В этом случае каждый $Stub_i$, содержащий n_i путей, разделенных на m_i классов, может использовать в подстановках не более m_i вариантов путей, по одному из каждого класса.

В результате оценка общего числа трасс для тестирования становится не более чем $N = \Pi m_i (i \in [1..n])$. Однако задача заключается в ограничении N до величины: $N = \Sigma n_i (i \in [1..n])$.

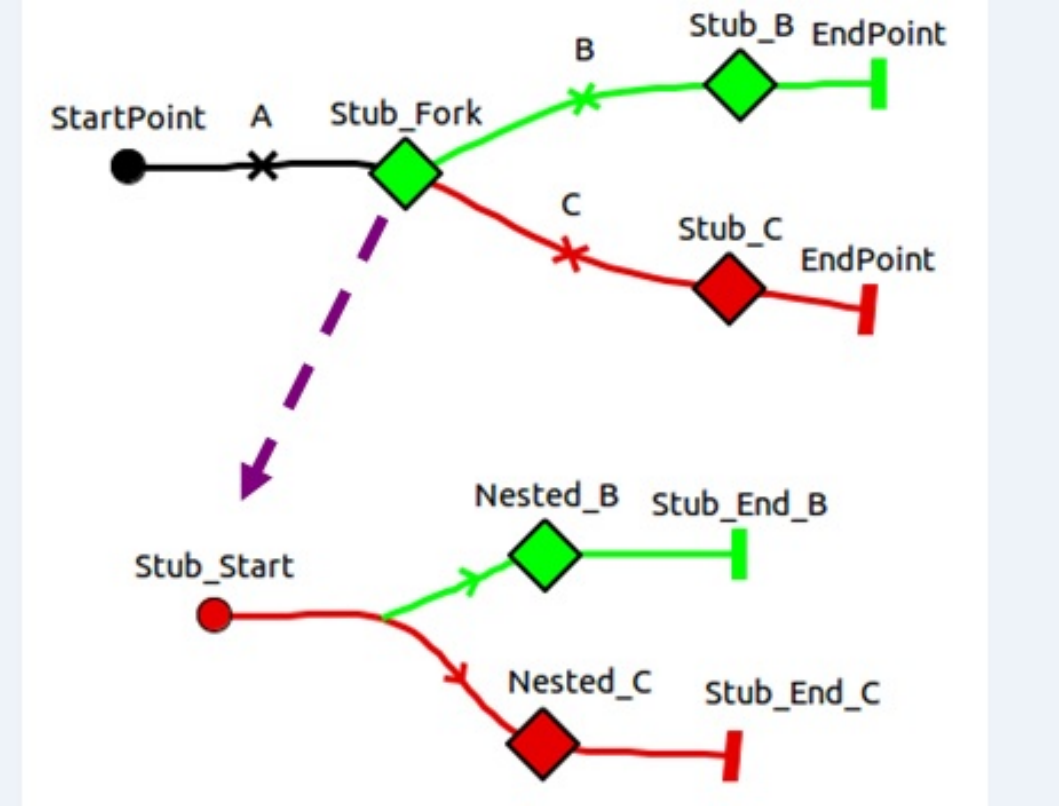

Рис. 4. Все пути через Stub Fork классифицируются на два класса – B и C

8. Метод оптимального отбора путей для склейки

Метод решает задачу оптимального отбора тестовых сценариев по критерию ветвей, число которых ограничено величиной

$$N = \Sigma n_i (i \in [1..n]) \tag{5}$$

за счет управления генерацией тестовых сценариев.

Пусть для некоторого проекта создана структурная модель – дерево компонент. Пусть в дереве автоматически сгенерированы пути, покрывающие поведение каждого Stub в отдельности. Пусть в дереве содержится n директорий для m элементов $Stub_i (i \in [1 \ldots n])$ и для каждой директории существует возможность выбора путей, используемых в процессе склейки. Кроме того, в каждой директории помечен путь (или несколько путей из разных классов), по умолчанию используемый при склейке. Тогда следующий процесс управления генерацией тестовых трасс решает задачу создания тестового набора.

1. Начинаем с диаграммы верхнего уровня, выбираем 1 из n_0 путей и генерируем трассу до первого Stub (т.е. до ссылки на Stub), а затем вклеиваем в него n_1 путей первого Stub. В результате получаем n_1 так называемых коротких трасс, заканчивающихся на первом Stub и обеспечивающих тестирование всех его ветвей (рис. 5а). Если трасса для генерации теста должна оканчиваться элементом EndPoint (так называемая длинная трасса), то соответству-

ющую короткую трассу необходимо продолжить до EndPoint, подставляя вместо включенных в путь ссылок на директории Stub содержащиеся в них пути по умолчанию. Полученные короткие или длинные трассы сохраняем в директории тестового набора.

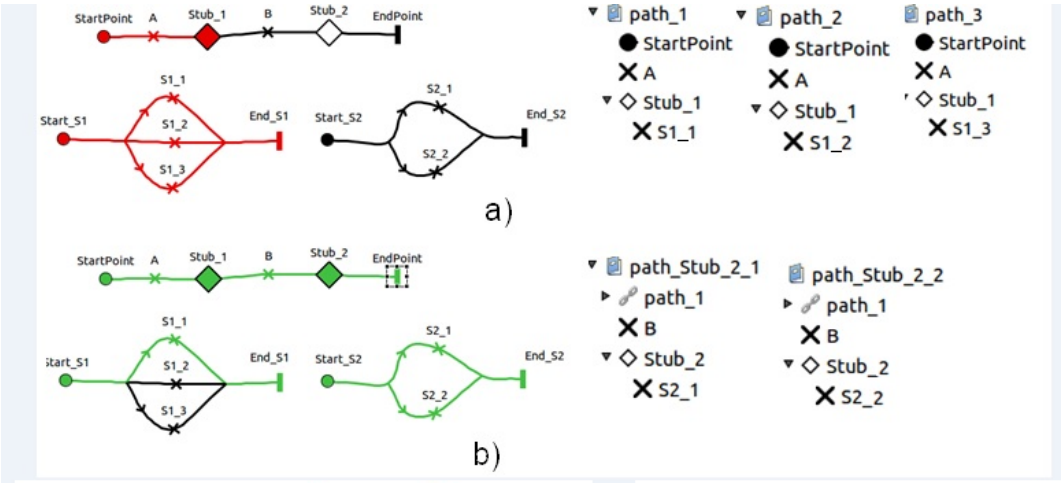

Рис. 5. a) Генерация $n_1 = 3$ трасс для $stub_1$; b) Генерация $n_2 = 3$ трасс для $stub_2$, с использованием прохождения $stub_1$ по умолчанию

2. Используем трассу с выбранным по умолчанию путем первого Stub и вклеиваем в него n_2 путей второго Stub. В результате получаем n_2 коротких трасс, обеспечивающих тестирование всех ветвей второго Stub (рис. 5b). Получение длинных трасс производится, аналогично подставляя вместо ссылок на Stub соответствующие пути по умолчанию. Полученные трассы добавляем к директории тестового набора. В результате в директории будет зафиксировано $n_1 + n_2$ трасс.

3. Повторяя процедуру для i Stub, получаем еще n_i трасс, обеспечивающих тестирование всех ветвей, покрытых n_i трассами. Трассы добавляем к директорию тестового набора, в которой будет зафиксировано $n_1 + n_2 + \cdots + n_i$ трасс $(i \in [1 \dots n])$.

4. Если путь, вклеиваемый в Stub верхнего уровня, в свою очередь содержит ссылки на Stub, то для него повторяется процедура п. 3 до тех пор, пока не будет достигнут путь, не содержащий ссылок на Stub.

5. Пункты с 1 по 4 необходимо повторить n_0 раз для путей, покрывающих диаграмму верхнего уровня.

9. Инструментарий

Описанный инкрементальный подход автоматизации проектирования тестовых трасс реализован в инструментальной системе VRS/TAT [9], в которой инструменты верификации, генерации тестовых наборов и собственно тестирования объединены в

единый автоматизированный процесс, требующий минимальных ручных усилий на фазах проектирования тестов и тестирования. Экспериментальная проверка эффективности инструментария и технологии его применения проводилась на телекоммуникационных приложениях, специфицированных моделями размера от 50 до 1500 базовых протоколов. В итоге применения всего технологического цикла VRS/TAT, от формализации требований и создания поведенческой модели до генерации тестовых наборов на целевую платформу, гарантирующих 100% покрытие исходных требований, трудоемкость фазы тестирования сократилась более чем на 30%. Причем трудоемкость корректировки исходной модели, происходящей в процессе разработки и требующей повторения этапов автоматизированного технологического цикла, осуществлялась с затратами, меньшими в 2–4 раза.

10. Заключение

Предложен подход к автоматизации инкрементального проектирования тестовых наборов для тестирования больших промышленных программных проектов. В процессе проектирования доказывается корректность поведенческой модели приложения и полнота покрытия тестовыми наборами функциональности исходных требований. Предлагаемый подход используется в едином технологическом цикле верификации и тестирования поведенческой модели, генерации по модели целевого кода тестового набора и обеспечения автоматического тестирования. Технология и инструментарий апробированы на проектах телекоммуникационных приложений.

Список литературы

1. Z.151 : User requirements notation (URN) — Language definition
`http://www.itu.int/rec/T-REC-Z.151-200811-I/en`

2. Crnkovic I., Larsson S., Chaudron M.R.V. Component-Based Development Process and Component Lifecycle // CIT. 2005. 13(4). P. 321–327.

3. Jisa, Laurentiu D. Component Based Development Methods // Comparison, Computer Systems and Technologies, 2004. P. 1–6.

4. Le H., Kathayat S.B. A Framework to Support the Development of Collaborative Components // 9th Workshop on System/Software Architectures. LNBIP, Springer, Heidelberg. 2011. Vol. 83. P. 378–384.

5. Sammi R., Rubab I., Qureshi M.A. Formal specification languages for real-time systems // Information Technology: International Symposium Proceedings (ITSIM-2010). 2010. Vol. 3. P. 1642–1647.

6. Kolano P.Z., Dang Z., and Kemmerer R.A. The Design and Analysis of Real-Time Systems Using the ASTRAL Software Development Environment // Annals of Software Engineering. 1999. Vol. 7. P. 177–210.

7. Greg Bollella. Ben Brosgol. Peter Dibble. Steve Furr. The Real-Time Specification for Java^{TM}. `www.cs.rice.edu/ taha/teaching/04F/RAP/cache`

8. Joss Warmer, Anneke. The Object Constraint Language, Precise Modeling with UML. Addison-Wesley, 1999.

9. Baranov S., Kotlyarov V., Letichevsky A., Drobintsev P. The technology of Automation Verification and Testing in Industrial Projects // Proc. of St. Petersburg IEEE Chapter, International Conference, May 18–21. St. Petersburg, 2005. P. 81–86.

10. Ануреев И.С., Баранов С.Н., Белоглазов Д.М., Бодин Е.В., Дробинцев П.Д., Колчин А.В., Котляров В.П., Летичевский А.А., Летичевский А.А. мл., Непомнящий В.А., Никифоров И.В., Потиенко С.В., Прийма Л.В., Тютин Б.В. Средства поддержки интегрированной технологии для анализа и верификации спецификаций телекоммуникационных приложений // Труды СПИИРАН. 2013. Вып. 3 (26). С. 349–383. [Anureev I.S., Baranov S.N., Beloglazov D.M., Bodin E.V., Drobintsev P.D., Kolchin A.V., Kotlyarov V.P., Letichevsky A.A., Letychevsky O.A., Nepomnyashchii V.A., Nikiforov I.V., Potiyenko S.V., Priima L.V., Tyutin B.V. Tools of Integrated Technology for Analysis and Verification of Telecom Application Specs // Tr. SPIIRAN. 2013. 3(26). P. 349–383 (in Russian)].

11. Hoare C.A.R. Communicating sequential processes. Prentice Hall, 1985.

12. Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky O.O., Volkov V.V., Baranov S.N., Weigert T. Basic Protocols, Message Sequence Charts, and the Verification of Requirements Specifications // Proc of ISSRE04 Workshop on Integrated Reliability Engineering (ISSRE04:WITUL). IRISA. Rennes France, 2004.

13. Колчин А.В. Разработка инструментальных средств для проверки формальных моделей асинхронных систем: Дис. ... канд. физ.-мат. наук. Киев, 2009. 140 с. [Kolchin A.V. Razrabotka instrumentalnykh sredstv dlya proverki formalnykh modeley asinkhronnykh sistem: Dis. ... kand. fiz.-mat. nauk. Kiev, 2009. 140 s. (in Russian)].

14. Baranov S., Kotlyarov V., Weigert T. Varifiable Coverage Criteria For Automated Tesdting. SDL2011: Integrating System and Software Modeling // LNCS. 2012. Vol. 7083. P. 79–89.

15. Letichevsky A.A., Kapitonova J.V., Kotlyarov V.P., Letichevsky A.A.,Jr., Nikitchenko N.S., Volkov V.A., and Weigert T. Insertion modeling in distributed system design // Software problems (Проблеми програмування). 2008. S. 13–38.

16. Месарович М., Такахара Я. Общая теория систем: математические основы. М.: Мир, 1978 [Mesarovich M., Takakhara Ya. Obshchaya teoriya sistem: matematicheskie osnovy. M.: Mir, 1978].

17. Drobintsev P., Kotlyarov V., Nikiforov I., Letichebsky A. A Formal Approachfor Generation of Test Scenarios Based on Guides // 5th Workshop “Program Semantics, Specification and Validation: Theory and Applications”. Yekaterinburg, Russia, June 24. 2013. P. 31–41.

18. Letichevsky A.A., Godlevsky A.B., Letichevsky Jr., A.A., Potienko S.V., Peschanenko V.S. Properties of Predicate Transformer of VRS System // Cybernetics and System Analyses. 2010. 4. P. 3–16.

19. Godlevsky A.B., Potienko S.V. Backward transformation of formulas in symbolic modeling: from the result to the source formula // Problems of Programming. 2010. 4. P. 363–368.

20. Dijkstra E.W., Scholten C.S. Predicate Calculus and Program Semantics. Springer-Verlag, 1990.

21. Drobintsev P.D., Nikiforov I.V., Kotlyarov V.P. Translation of UCM Real-Time Constructs into Basic Protocols // University Journal. 2013. №5. P. 193–201.

Incremental Approach to the Technology of Test Design for Industrial Projects

Drobintsev P.D.[1], Kotlyarov V.P.[1], Nikiforov I.V.[1], Letichevsky A.A.[2]

[1] *St. Petersburg State Polytechnical University*
Polytechnicheskaya st., 29, St. Petersburg, 195251, Russia
[2] *Glushkov Institute of Cybernetic of NAS of Ukraine,*
Glushkova av., 40, Kyiv, 03187, Ukraine

Keywords: model driven testing, integrated verification and testing, automation design of test scenarios, reducing of test explosion

The paper presents an approach to effort reduction in developing test suites for industrial software products based on the incremental technology. The main problems to be solved by the incremental technology are full automation design of test scenarios and significant reducing of test explosion. The proposed approach provides solutions to the mentioned problems through joint co-working of a designer and a customer, through the integration of symbolic verification with the automatic generation of test suites; through the usage of an efficient technology with the toolset VRS/TAT

Сведения об авторах:
Дробинцев Павел Дмитриевич,
Санкт-Петербургский государственный политехнический университет,
доцент, канд. техн. наук;
Котляров Всеволод Павлович,
Санкт-Петербургский государственный политехнический университет,
профессор, канд. техн. наук;
Никифоров Игорь Валерьевич
Санкт-Петербургский государственный политехнический университет, аспирант;
Летичевский Александр Адольфович,
Институт кибернетики им. В.М. Глушкова НАН Украины,
зав. отделом теории цифровых автоматов, академик, д-р физ.-мат. наук