УДК 004.4'24 + 004.415.5 + 004.415.28

А.В. Колчин, В.П. Котляров, П.Д. Дробинцев

Метод генерации тестовых сценариев в среде инсерционного моделирования

Описан метод поиска специфических поведений формальных моделей, использующий задаваемую пользователем дополнительную информацию о поведенческих особенностях проектируемой системы, и его применение к автоматическому построению тестовых сценариев для программных систем.

A method of the search of specific behaviours of the formal models which uses an additional information about peculiarities of the progected system and its application to the automatic construction of test scenarious for program systems is desceibed.

Описано метод пошуку специфічних поведінок формальних моделей, за яким використовується задавана користувачем додаткова інформація про поведінкові особливості проектованої системи та його застосування у автоматичній побудові тестових сценаріїв для програмних систем.

Введение. Экспериментальная проверка соответствия требований и реализации программного продукта обеспечивается тестированием. Поскольку проверка поведения программы во всех ситуациях практически невозможна, прибегают к определению критериев покрытия и ограничиваются соответствующим набором тестовых сценариев. При этом, как правило, полнота тестового покрытия оценивается по числу выполненных операторов, ветвей, путей, достижению граничных значений функций, выполнению предикатов модели и т.д. [1]. Подобные критерии удобно использовать для автоматической генерации тестов, которые могут обеспечить покрытие кода разрабатываемого продукта, но не требований к нему. Например, из того, что тестовый набор покрывает все предикаты и операторы модели, не следует, что модель правильно реагирует на входящие сигналы.

Обычно исходные требования, формулируемые заказчиком, и их детальные спецификации, создаваемые разработчиками, заданы на разных уровнях детализации. Для того, чтобы автоматизировать проверку соответствия спецификаций разработчиков требованиям заказчика, необходимо построить формальные модели спецификаций, согласованные с требованиями заказчика и их интерпретацией разработчиком. Как правило, последние представляют собой сценарии типичных режимов поведения (*use-cases*) разрабатываемой системы. Такие сценарии можно переформулировать в виде последовательностей (цепочек) событий модели, успешное прохождение (выполнение) которых будет означать покрытие соответствующего требования [2, 3]. Подобные цепочки событий, соотнесенные с требованием, называют целью теста (*test purpose*).

Разработка тестового набора, обеспечивающего покрытие требований к продукту, достаточно сложна и вполне сопоставима с трудоемкостью создания кода. Для снижения трудоемкости создания тестов активно применяются методы и системы автоматической генерации тестовых сценариев на основе формальных моделей разрабатываемых систем (*model based testing*) [4]. Хотя получаемые в результате тесты обычно слабо связаны со специфическими особенностями кода тестируемой системы, тем не менее, они содержат представительный набор сценариев поведения системы, что позволяет значительно сократить трудозатраты тестирования сложных систем. Обзоры таких работ можно найти в [1, 4–8]. В работах [9, 10], помимо формальной модели, на вход подается сценарий тестирования, заданный пользователем в виде последовательности сообщений, которыми обмениваются компоненты модели, например, в виде *MSC* [10, 11]. К проблемам такого подхода можно отнести недостаточную выразительную мощность языка описания свойств. Системы верификации [12, 13] предполагают,

что свойство, которое необходимо проверить на модели, задано в виде некоторой формулы темпоральной логики. Этим можно было бы воспользоваться, задав такую формулу, которая станет ложной на некотором пути. Тогда такой путь будет выдан верификационной системой в качестве контрпримера и будет рассматриваться как один из сценариев покрытия некоторого требования. Однако на практике нет однозначного соответствия между формулой над атрибутами модели и желаемой цепочкой событий (обусловленной исходными требованиями).

Таким образом, актуальной остается проблема семантического соответствия между генерируемыми тестовыми сценариями и критериями покрытия исходных функциональных требований к программному продукту.

При генерации тестовых сценариев также актуальна проблема комбинаторного взрыва количества состояний формальной модели. Решению проблемы посвящено множество активно развивающихся эвристических методов управления поиском (например [14, 15]) и редукции пространства анализируемых состояний модели. Особенно востребованы эффективные методы элиминации избыточного интерливинга. Действительно, количество возможных трасс модели, состоящей из p процессов, каждый из которых независимо друг от друга последовательно выполняет n атомарных действий, определяется формулой $\frac{(n \cdot p)!}{(n!)^p}$. Таким образом, в модели, состоящей всего из трех процессов, выполняющих по шесть действий, количество трасс превышает 17 млн. При этом современные методы эффективно работают далеко не всегда (например, из-за уязвимости к зависимостям, обусловленным недостижимыми переходами, и т.д. [16]).

Метод направленного поиска

В работах [8, 17, 18] описан метод направленного поиска, предназначенный для решения задачи генерации тестовых сценариев, удовлетворяющих заданным критериям покрытия. Используемые критерии покрытия (гиды, *Guides*) формулируются в виде специальных регулярных выражений над алфавитом имен переходов модели. Гиды абстрактно определяют искомые поведения в терминах событий модели и одновременно накладывают ограничения на множество анализируемых состояний, путем отсечения ветвей поведения, не удовлетворяющих заданным критериям. Данная статья расширяет этот метод, а также описывает опыт его применения в промышленных проектах.

Далее представлены формальные определения модели и языка гидов.

Определение 1. Транзиционной системой M называется пятерка $<Q, q_0, T, P, f>$, где Q – множество состояний, $q_0 \in Q$ – начальное состояние, T – множество имен параметризированных переходов, P – (динамическое) множество агентов, а $f{:}Q \to P$ – отображение, указывающее фактическое множество агентов в состоянии Q.

Для простоты описания мы ассоциируем события модели с именами ее переходов. Параметризация переходов актуальна в особенности для распределенных систем, состоящих из параллельных асинхронных процессов, которые могут порождаться и удаляться динамически. Агентом может быть абстрактно представлен один или множество процессов [19].

Определение 2. Путем в M из состояния q_i в состояние q_j называется такая последовательность состояний и переходов $q_i \xrightarrow{t_i(a_i)} q_{i+1} \times \times \xrightarrow{t_{i+1}(a_{i+1})} q_{i+2} \ldots q_j$, что $q_k \in Q \wedge t_k \in T \wedge a_k \in f(q_k)$ для всех $k \in i \ldots j$.

Определение 3. Трассой в M называется последовательность $t_0(a_0), t_1(a_1), \ldots, t_n(a_n) \ldots$ такая, что существует путь $q_0 \xrightarrow{t_0(a_0)} q_1 \xrightarrow{t_1(a_1)} \ldots \ldots \xrightarrow{t_n(a_n)} q_n \ldots$

Определение 4. Язык, ассоциированный с M, обозначается $\mathcal{L}(M) \subset T^*$ – это набор всех трасс, выходящих из начального состояния q_0.

Пусть заданы непересекающиеся алфавиты X и Y, а так же язык Z над X. Пусть задано отображение $\lambda : X \to X \cup Y$, определенное как $\lambda(x) = \{AxB \mid A, B \in Y^*\}$. Тогда язык $Z_{\Uparrow Y} =$

$= \{\lambda(\gamma) | \gamma \in Z\}$ над алфавитом $X \cup Y$ является расширением языка Z до алфавита Y.

Определение 5. Гидом называется

$a.n$ – переход a на максимальной дистанции n, который допускает множество трасс $\{a, X_1 a, \ldots, X_1 \ldots X_n a\}$, где $X_1, \ldots, X_n$ – любые непустые символы из $\{T \backslash a\}$;

$\sim a$ – запрет перехода a, допускает любой символ из $\{T \backslash a\}$;

a; b – (где a, b – гиды) конкатенация гидов, допускает множество трасс $\{ab\}$;

$a \vee b$ – (где a, b гиды) недетерминированный выбор гидов, допускает множество трасс $\{a, b\}$;

$a \parallel b$ – параллельная композиция гида a, представляющего язык Z^a над алфавитом множества X, и b, представляющего язык Z^b над алфавитом множества Y, причем $X \cap Y = \varnothing$, так как множества агентов в X и Y не пересекаются. Параллельная композиция гидов – это множество, представленное языком $Z^{a \| b} = Z^a_{\Uparrow Y} \cap Z^b_{\Uparrow X}$;

$join(a_1, \ldots, a_n)$ – множество $\{S_n\}$ всех перестановок гидов $a_1, \ldots, a_n$;

$loop(a)$ – итерация гида a, т.е. $\{aa^*\}$.

Свойства приведенных операций подробно описаны в [8].

Переходы, входящие в гид, называются наблюдаемыми. Два наблюдаемых перехода на трассе расположены на дистанции d, если между ними расположены d−1 переходов. Дистанция первого перехода в гиде есть расстояние от начального состояния. В инструменте поддержки метода [20] дистанции в гидах вычисляются автоматически или задаются вручную в случае ограничения итераций или рекурсий.

Запреты переходов действуют до обнаружения следующего допустимого наблюдаемого перехода и не могут быть последними в гиде (за исключением использования внутри итерации).

Параллельная композиция и оператор *join* используются для случаев, когда модель имеет несколько параллельно работающих компонент (процессов), при этом дистанция учитывается для каждого параллельного участка гида отдельно.

Для обнаружения бесконечных циклов используется оператор *loop*. С его помощью, например, можно проверить, приходит ли модель системы в устойчивое состояние при отсутствии внешних сигналов. Пусть E_1 и E_2 – единственные воздействия (переходы) внешней среды, а A – переход некоторого единственного процесса *ms*. Пусть так же задан гид: $E_1.2$; $A.4$; $loop(\sim E_1 \backslash / \sim E_2)$. Тогда трасса, удовлетворяющая гиду, должна иметь цикл, в котором нет переходов E_1 и E_2, а префикс этой трассы должен включать переходы E_1 и A на соответствующих дистанциях. Существование такой трассы означает, что после воздействия среды событием, размеченным переходом E_1 и последующей реакцией системы переходом A, процесс *ms* может не прийти в устойчивое состояние (в цикл не входит ни одно событие от внешней среды).

Так как переходы параметризированы агентом, а множество агентов в трассе может изменяться динамически, вводится параметризация гидов, при этом в процессе порождения трасс в каждом новом состоянии проверяется существование соответствующих подстановок. Таким образом, построенная трасса удовлетворяет гиду, если она содержит наблюдаемые переходы в непротиворечащей заданному гиду последовательности, на дистанциях, не превышающих указанные, и существует соответствие параметров гида агентам в трассе. Например, гид

$$A.2;\ (B.2 \vee (\sim Z, C.4))$$

имеет примеры удовлетворяющих трасс:

$$X, A, Z, B$$

$$A, X, C$$

пример не удовлетворяющей трассы:

$$A, X, Z, C.$$

Как видно, выражения, не содержащие циклов, всегда имеют ограничения на длину трассы, удовлетворяющей гиду; более того, определить, что трасса не удовлетворяет гиду, можно не достигая максимально допустимой длины, так как всегда определена максимальная дистанция между событиями. В приведенном примере максимальная длина трассы 6 ($A.2$, $C.4$), но (A, Z, C), имея длину три, уже может быть прервана.

Семантически такими гидами задаются «контрольные точки» поведения модели и так же

определяется критерий (цепочка событий в поведении модели) выбора построенных трасс для последующего их использования в качестве тестовых сценариев [2, 3, 21]. Так, предполагаемые поведения моделируемой системы проверяются на допустимость, одновременно накладывая ограничения на их поиск.

Реализация метода

По каждому гиду строится соответствующий детерминированный конечный автомат. Реализованы точный и жадный алгоритмы распознавания для соответствующих типов регулярных выражений [22]. Важно отметить, что на практике удовлетворительный результат можно получить, используя жадные алгоритмы линейной сложности. Из соображений эффективности, реализация метода позволяет принимать на вход несколько гидов, при этом осуществляется множественный поиск трасс одновременно всего набора гидов, а также вводятся ограничители на число необходимых трасс по каждому гиду. Обход пространства поведения, реализующий адаптированный алгоритм Тарьяна [23], модифицируется путем добавления перед шагом вглубь вызова специальной функции, реализующей автомат, распознающий гиды, при этом *полным* состоянием будет синхронное произведение состояний модели (значений атрибутов) и состояний автомата, распознающего гиды. Если текущая трасса удовлетворяет гиду, она будет сохранена на диск, если не исчерпан соответствующий лимит (пользователь предварительно определяет нужное количество трасс по каждому гиду). Если лимит исчерпан, гид исключается из списка актуальных. Если данная функция возвращает на выходе `false`, это означает, что достигнута ситуация, из которой любое продолжение текущей трассы не приведет к обнаружению вхождения очередного символа (т.е. имени наблюдаемого перехода) хотя бы одного актуального гида. Такая ситуация является дополнительным критерием завершения генерации текущей трассы. При этом, если трасса включает ранее не покрытый символ гида, она будет сохранена на диск с соответствующей пометкой. Таким образом, если полного вхождения какого-либо гида не обнаружено, будут построены трассы частичного покрытия. Дополнительным для сохранения трассы есть условие вхождения в трассу перехода, не входящего ни в одну из уже сохраненных трасс. В итоге будет построен набор трасс, включающий как предполагаемые пользователем поведения модели, так и все достигнутые переходы модели.

Опыт использования метода

Метод направленного поиска был встроен в трассовый генератор системы *VRS* [19, 24], активно применялся [2, 3, 20, 21, 25–29] и развивался [8, 17, 18] на протяжении последних пяти лет. Далее представлены некоторые аспекты его использования для решения задач достижимости и генерации тестов.

Сокращение пространства поиска

Применение метода в реальных индустриальных проектах выявило несколько проблем, не очевидных с теоретических позиций. Так, при множественном поиске в случае, когда гиды описывают трассы, находящиеся на удаленных ветвях обхода поведения модели, хранилище пройденных состояний становится большим и, как следствие, поиск заметно замедляется. Для таких случаев эффективным оказалось разбиение исходного множества гидов на подмножества и проведение отдельных экспериментов.

Другой проблемой становились завышенные оценки дистанций между переходами. Для таких случаев эффективнее применялся поиск в ширину и режим использования жадного алгоритма.

Во избежание ненужных с точки зрения тестового сценария (или проверяемых свойств) перестановок, эффективным оказалось использование оператора *join*, который допускает лишь перестановки аргументов, но не интерливинг между ними.

Генерация тестовых сценариев

В практике проектирования качественных программных продуктов разработчикам необходимо согласовать с заказчиком семантику требований и формализовать процедуру их тестового покрытия. Опыт показывает, что наиболее удобной и понятной для обеих сторон запись такого согласования выражается в терминах событий исходной модели. Так, для каждого требования на языке описания гидов формулиро-

вались специальные конструктивные критерии [3], которые, с одной стороны, подтверждали семантическое соответствие требованию, с другой, служили дополнительным входом верификатора для автоматической генерации трасс.

В работах [25, 29] описано применение предложенного метода к процессу генерации тестовых сценариев на базе двух формализованных моделей: одной – в виде высокоуровневой нотации *UCM*-диаграмм (*Use Case Maps*), используемой для контролируемого заказчиком описания поведения и согласования с ним поведенческих сценариев, другой – в нотации транзиционных систем, детально описывающей переходы и события модели. Разработанные методы были интегрированы в технологию *VRS*/*TAT*, где обеспечили 30–40% сокращения трудоемкости проектирования телекоммуникационных приложений. Например, в работе [21] описано применение метода направленного поиска для верификации и тестирования телекоммуникационного проекта. В результате было сгенерировано 1214 тестовых сценариев, из которых было отобрано 196, обеспечивших 100% покрытие требований. В процессе верификации было найдено 32 случая некорректности: шесть тупиков (не детализированные фрагменты поведения в требованиях), 13 противоречий между кодом и требованиями, семь противоречий в документации, шесть некорректностей других типов.

Далее представлена таблица сводных данных, полученных в индустриальных проектах с использованием предлагаемого подхода к тестированию.

Таблица. Статистика применения описанного подхода в индустриальных телекоммуникационных проектах

Проект	Количество требований	Трудоемкость тестирования (чел./недели)		% сокращения трудозатрат
		С применением технологии	Традиционный	
# 1	107	1,8	5,9	69
# 2	148	2,2	5,7	61
# 3	51	2,1	6,3	67
# 4	57	2,8	7	60
# 5	730	6	~18	~72
# 6	1452	12	~40	~70

Примеры проектов большого размера – # 5 и # 6. Для них создавались тестовые сценарии только по предложенному методу; данные для традиционного подхода – оценочны.

Опыт, полученный в приведенных проектах, показал, что для обеспечения полного покрытия требований тестами эффективно использовать итеративный процесс генерации тестовых сценариев из формальной модели, основанный на применении гидов. Также стало понятно, что можно значительно улучшить качественные характеристики гидов и автоматизировать процесс их генерации с помощью дополнительной информации, вводимой пользователем вручную. Так, важной информацией от пользователя является [3] указание (абстрактных) путей в *UCM* диаграмме, которые не покрываются автоматически с использованием структурного критерия. Такие пути автоматически транслируются в соответствующие гиды [29], после чего покрывающие трассы генерируются верификатором.

Рассмотренный подход на практике позволяет решить проблему экспоненциального взрыва количества состояний при генерации тестовых сценариев за счет использования структурирования и автоматизации покрытия ветвей и путей. Например, для телекоммуникационного проекта из 148 требований удалось сократить число покрывающих трасс с 2^{150} до 2^5. Определение несоответствия трассы критериям покрытия определяется оперативно, в процессе обхода пространства поведения модели. Эта особенность отличает его от подхода, описанного в [30] и использующего созданные вручную тестовые шаблоны.

Важная особенность предлагаемого подхода – частое взаимодействие с заказчиком для определения семантики требований. После утверждения заказчиком набора последовательностей событий покрывающих, с его точки зрения, то или иное требование, заказчик разделяет ответственность за правильность конечного тестового набора, а генерация тестов становится чисто технической задачей, которая может быть автоматизирована.

Заключение. Описанный метод направленного поиска – разновидность ограниченной проверки модели (*bounded model checking*), в качестве ограничителей использует цели тестирова-

ния, определяемые пользователем. Метод эффективен в качестве эвристики при решении задач достижимости [21, 26]. В отличие от существующих методов управления поиском и генерации тестов, направлением для поиска служит гид – специальное ограниченное регулярное выражение над алфавитом имен переходов модели. Такой гид служит прообразом тестового сценария, используется для отсечения несовместимых ветвей поведения модели и одновременно для проверки предполагаемого поведения на допустимость, при этом, по сути, осуществляется валидация модели. Метод позволяет эффективно находить труднодостижимые состояния, а также проверять нарушение специфических условий безопасности – например, наличие бесконечных циклов в поведении процесса при отсутствии внешних сигналов.

Технология применения метода направленного поиска использует вход и выход в терминах исходной модели – проверяемые свойства формулируются в виде регулярных выражений над событиями модели, а результирующие трассы представляют собой линейную последовательность таких событий.

Многие существующие инструменты [6, 9] выполняют поиск с фиксированным ограничением на длину трассы, и как следствие, отсутствие результата не означает, что модель не допускает искомого тестового сценария. Описанный метод фиксирует локальные в рамках трассы ограничения на тестовый сценарий и при этом гарантирует обнаружение удовлетворяющих трасс.

Необходимо отметить, что метод направленного поиска может быть использован совместно с методами абстракции предикатов, частичного порядка и многими другими методами редукции анализируемого пространства поведения модели.

Применение технологии направленного поиска в промышленных проектах дало возможность не только выявить ошибки в спецификациях, но и построить набор тестовых сценариев, обеспечивающих необходимое функциональное покрытие [3, 21, 26] при условии сохранения семантики требований.

Данная разработка поддержана грантом РФФИ 11-07-90412-Укр_ф_а (Россия), научно-исследовательской работой № Ф40\51-2011 (Украина) и ДФФД по проекту Ф40.1/004.

1. *Beizer B*. Software Testing Techniques. ITP. – 1990. – 550 p.
2. *Котляров В.П.* Критерии покрытия требований в тестовых сценариях, сгенерированных из поведенческих моделей приложений // Науч.-техн. ведомости СПбГПУ. – 2011. – Т. 6.1(138). – С. 202–207.
3. *Baranov S*., *Kotlyarov V*., *Weigert T*. Varifiable Coverage Criteria For Automated Testing. SDL2011: Integrating System and Software Modeling // LNCS. – 2012. – **7083**. – P. 79–89.
4. *Utting M*., *Pretschner A*., *Legeard B*. A Taxonomy of Model-Based Testing // Techn. Report, Depart. of Comp. Sci., The University of Waikato, New Zealand. – 2006. – 17 p.
5. *Кулямин В.В.* Методы верификации программного обеспечения // Всерос. конкурсный отбор обзорно-аналитических статей по приоритетному направлению «Информационно-телекоммуникационные системы». – 2008. – 117 с.
6. *Обзор* подходов к верификации распределенных систем / И.Б. Бурдонов, А.С. Косачев, В.Н. Пономаренко и др. – М.: ИСП РАН, 2006. – 61 с.
7. *IBM*: Лучшие средства разработки программного обеспечения. – http://www-01.ibm.com/software/ru/rational/software_development
8. *Летичевский А.А*., *Колчин А.В*. Генерация тестовых сценариев на основе формальной модели // Проблемы программирования. – 2010. – № 2–3. – С. 209–215.
9. *Utting M*., *Legeard B*. Practical Model-Based Testing: A Tools Approach. Morgan-Kaufmann. – 2007. – 456 p.
10. *BZ-TT*: A tool-set for test generation from Z – and B – using constraint logic progr. / F. Ambert, F. Bouquet, S. Chemin et al. // Proc. of FATES'2002. – 2002. – P. 105–119.
11. *Farchi E*., *Hartman A*., *Pinter S*. Using a model-based test generator to test for standard conformance // IBM Syst. J. – 2002. – **41**(1). – P. 89–110.
12. *Ben-Ari M*. Principles of Spin // Springer Verlag, 2008. – 216 p.
13. *NuSMV* 2: An OpenSource Tool for Symbolic Model Checking / A. Cimatti, E. Clarke, E. Giunchiglia et al. // Proc. of Int. Conf. on Computer-Aided Verification, Copenhagen, Denmark. – 2002. – P. 359–364.
14. *Edelkamp S*., *Leue S*., *Lafuente A*. Directed explicit-state model checking in the validation of communication protocols // Int. J. on software tools for technology transfer. – 2003. – N 5. – P. 247–267.
15. *Lafuente A*. Directed Search for the Verification of communication protocols // Doctorial thesis, Institute of computer science, University of Freiburg. – 2003. – 157 p.
16. *Колчин А.В*. Автоматический метод динамического построения абстракций состояний формальной модели // Кибернетика и системный анализ. – 2010. – № 4. – С. 70–90.

Окончание на стр. 63

17. *Колчин А.В.* Метод направления поиска и генерации тестовых сценариев при верификации формальных моделей асинхронных систем // Проблемы программирования. – 2008. – № 4. – С. 5–12.
18. *Колчин А.В.* Направленный поиск в верификации формальных моделей // Тези доп. Міжнар. конф. «Теоретичні та прикладні аспекти побудови програмних систем *TAAPSD*'2007» – Бердянск. НаУКМА, Нац. ун-т імені Тараса Шевченка, Ін-т програмних систем НАН України. – 2007. – С. 256–258.
19. *Insertion* modeling in distributed system design / A.A. Letichevsky, Ju.V. Kapitonova, V.P. Kotlyarov et al. // Проблеми програмування. – 2008. – С. 13–38.
20. *Дробинцев П.Д.*, *Котляров В.П.*, *Черноруцкий И.Г.* Автоматизация тестирования на основе покрытия пользовательских сценариев // Научно-технические ведомости СПб ГПУ. – 2012. – № 4 (152). – 10 с.
21. *Баранов С.Н.*, *Котляров В.П.*, *Летичевский А.А.* Индустриальная технология автоматизации тестирования мобильных устройств на основе верифицированных поведенческих моделей проектных спецификаций требований // Труды Междунар. науч. конф. «Космос, астрономия и программирование» – СПб ГПУ. – 2008. – С. 134–145.
22. *Smyth B.* Computing Patterns in Strings // ACM Press Books. – 2003. – 440 p.
23. *Tarjan R.* Depth first search and linear graph algorithms // SIAM J. on Computing. – 1972. – **1**(2). – P. 146–160.
24. *Свойства* предикатного трансформера системы *VRS* / А.А. Летичевский, А.Б. Годлевский, А.А. Летичевский (мл.) // Кибернетика и системный анализ. – 2010. – № 4. – С. 3–16.
25. *Котляров В.П.*, *Дробинцев П.Д.* Автоматизация проектирования программного продукта с помощью формальных спецификаций. – Вест. НГУ. Серия: Информационные технологии. – Новосибирск. – 2011. – Т. 9(4). – С. 29–38.
26. *Воинов Н.В.*, *Котляров В.П.* Применение метода эвристик для создания оптимального набора тестовых сценариев // Науч.-техн. вед. СПб ГПУ. – 2010. – № 103. – С. 169–174.
27. *Дробинцев П.Д.*, *Ким Р.И.*, *Котляров В.П.* Автоматизация тестирования с использованием символических трасс // Там же. – 2011. – № 126. – С. 175–180.
28. *Баранов С.Н.*, *Котляров В.П.* Формальная модель требований, используемая в процессе генерации кода приложения и кода тестов // Моделирование и анализ информационных систем. – Ярославский гос. ун-т им. П.Г. Демидова. – 2011. – Т. 18(4). – С. 118–130.
29. *Воинов Н.В.* Методы генерации тестовых сценариев на основе структурированных *UCM*-моделей проектируемой системы: Дис. … канд. техн. наук. – СПб. – 2011. – 169 с.
30. *Hassine J.*, *Rilling J.*, *Dssouli R.* Use Case Maps as a property specification language. Software and Systems Modeling. – 2009. – **8**(2). – P. 205–220.

Тел. для справок: +38 044 200-8423 (Киев)
E-mail: *kolchin_av@yahoo.com*, *vpk@spbstu.ru*