На правах рукописи

СТЕПАНОВ Олег Георгиевич

Методы реализации автоматных объектно-ориентированных программ

Специальность 05.13.11. Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

АВТОРЕФЕРАТ
диссертации на соискание ученой степени
кандидата технических наук

Санкт-Петербург
2009

Работа выполнена в Санкт-Петербургском государственном университете информационных технологий, механики и оптики (СПбГУ ИТМО)

Научный руководитель:	доктор технических наук, профессор Шалыто Анатолий Абрамович
Официальные оппоненты:	доктор физ.-мат. наук, профессор Романовский Иосиф Владимирович
	кандидат технических наук Нарвский Андрей Сергеевич
Ведущая организация	Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

Защита диссертации состоится **24 декабря 2009** года в $\mathbf{15}^{\underline{\mathbf{30}}}$ на заседании диссертационного совета Д212.227.06 при Санкт-Петербургском государственном университете информационных технологий, механики и оптики, 197101, Санкт-Петербург, ул. Кронверкский 49.

С диссертацией можно ознакомиться в библиотеке СПбГУ ИТМО.

Автореферат разослан **23 ноября 2009 г.**

Ученый секретарь диссертационного совета,
доктор технических наук, доцент Л.С. Лисицына

Общая характеристика работы

Актуальность темы. В настоящее время актуальной является проблема разработки качественного программного обеспечения (ПО). При разработке такого ПО важно учитывать различные аспекты качества, одним из которых является *корректность*: соответствие реализации программы заданной спецификации. Важно отметить, что в современных проектах по разработке ПО изменение спецификации даже внутри цикла разработки – обычное явление, вызванное уточнением понимания предметной области и изменением внешних условий. Другой особенностью современных программных проектов является регулярная перестройка исходного кода программы для облегчения поддержки изменений в спецификации и улучшения читаемости. Такая перестройка, при которой сохраняется поведение программы, называется рефакторингом.

Существующие технологии поддержания качества программных продуктов обладают рядом недостатков, основным из которых является то, что эти технологии основаны на тестировании. Тестирование не может дать гарантию отсутствия ошибок в программе. Технологии, основанные на верификации (формализованной проверке соответствия программы спецификации), в настоящее время недостаточно эффективны для использования в программах, написанных традиционным способом.

Эти недостатки сказываются на качестве модулей программ со сложным поведением, для которых тестирование оказывается наименее эффективным. В модулях со сложным поведением реакция на вызов зависит от внутреннего состояния модуля. При этом отметим, что в России с 1991 года разрабатывается концепция автоматного программирования, которая позволяет эффективно разрабатывать модули со сложным поведением. Частью этой концепции являются технологии совмещения автоматного и объектно-ориентированного программирования. Это позволяет реализовывать модули со сложным поведением с использованием автоматного программирования в различных проектах, включая уже существующие.

Автоматные программы, в отличие от программ традиционного типа, могут быть эффективно верифицированы с помощью метода *Model Checking*, так как в таких программах первичными являются модели, а в традиционных – код. Однако в настоящее время не разработаны методы, позволяющие интегрировать технологии реализации автоматных программ и их верификации в современные процессы разработки объектно-ориентированного ПО. В частности, недостаточно проработаны следующие вопросы:

- механизм эффективной формализации требований к автоматным программам;

- интеграция спецификации и исполняемого кода автоматных программ с учетом его верификации;
- синхронизация требований и реализации автоматных программ (односторонняя).

Исходя из этого, можно утверждать, что исследования, направленные на разработку методов реализации качественных автоматных объектно-ориентированных программ, весьма актуальны.

Целью диссертационной работы является создание методов реализации автоматных объектно-ориентированных программ, позволяющих интегрировать процессы разработки автоматных и объектно-ориентированных программ.

Основными задачами исследования является создание:

- метода формализации требований к автоматным объектно-ориентированным программам с использованием контрактов;
- метода интеграции спецификации и реализации автоматных объектно-ориентированных программ;
- метода модификации автоматных программ, уменьшающего число изменений, которые могут привести к появлению ошибок;
- каталога рефакторингов.

Методы исследования. В работе использованы методы теории автоматов, верификации, контрактного программирования и объектно-ориентированного проектирования.

Научная новизна. В работе получены следующие научные результаты, которые выносятся на защиту:

- метод формализации требований к автоматным объектно-ориентированным программам в виде формул темпоральной логики и контрактов, применяемых при статической и динамической верификации;
- метод интеграции формализованных требований и кода автоматных объектно-ориентированных программ на основе использования возможностей мультиязыковой среды программирования;
- метод модификации автоматных программ, основанный на рефакторинге, уменьшающий число изменений, которые могут привести к появлению ошибок;

- каталог рефакторингов автоматных программ, и доказательство эквивалентности программ до и после рефакторинга.

Достоверность научных положений, выводов и практических рекомендаций, полученных в диссертации, подтверждается корректным обоснованием постановок задач, точной формулировкой критериев, компьютерным моделированием, а также результатами внедрения методов, предложенных в диссертации, на практике.

Практическое значение работы состоит в том, что все полученные результаты могут быть использованы, а некоторые уже используются, в практической деятельности компании *ООО «ИнтеллиДжей Лабс»* (Санкт-Петербург). Предложенные методы реализации упрощают разработку, поддержку и сопровождения автоматных объектно-ориентированных программ за счет интеграции спецификации и кода программы.

Внедрение результатов работы. Результаты, полученные в диссертации, используются на практике:

- в компании *ООО «ИнтеллиДжей Лабс»* при разработке системы учета дефектов *YouTrack*.
- в учебном процессе по курсу «Применение автоматов в программировании» кафедры «Компьютерные технологии» СПбГУ ИТМО.

Апробация диссертации. Основные положения диссертационной работы докладывались на III Межвузовской конференции молодых ученых (СПб., 2006 г.), XXXVI научной и учебно-методической конференции профессорско-преподавательского и научного состава СПбГУ ИТМО (2007 г.), конференциях Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE) (СПбГУ, 2008 г.), 5th Central and Eastern European Software Engineering Conference in Russia (SECR) (М., 2009 г.).

Публикации. По теме диссертации опубликовано четыре статьи, в том числе две в журналах из перечня ВАК. В работах, выполненных в соавторстве, автором получено не менее половины результатов.

Гранты. Диссертация выполнялась в ходе работы над грантом для студентов, аспирантов вузов и академических институтов, находящихся на территории Санкт-Петербурга, который проводился Администрацией Санкт-Петербурга (2008 г.). Материалы диссертации используются в научно-исследовательской работе по теме «Методы повышения качества при разработке автоматных программ с использованием функциональных и объектно-ориентированных языков программирования», которая победила в конкурсе НК-421П «Проведение научных исследований научными группами под руководством кандидатов наук» по направлению «Информатика», который был

объявлен в 2009 году Федеральным агентством по образованию по Федеральной целевой программе «Научные и научно-педагогические кадры инновационной России» на 2009–2013 годы.

Структура диссертации. Диссертация изложена на 153 страницах и состоит из введения, пяти глав и заключения. Список литературы содержит 69 наименований. Работа содержит 23 рисунка и одну таблицу.

Содержание работы

Во введении обосновывается актуальность темы диссертационной работы, описываются цель и задачи исследования. Формулируются положения, выносимые на защиту.

Первая глава содержит обзор существующих методов разработки объектно-ориентированных программ, методов совмещения автоматного и объектно-ориентированного программирования, а также методов повышения качества автоматных программ.

Одной из особенностей создания объектно-ориентированного программного обеспечения является возможность изменения требований во время разработки программы. Для обеспечения готовности к таким изменениям существует ряд специальных технологий, такие так гибкие процессы разработки, рефакторинг и программирование по контрактам.

Основные технологии проверки качества объектно-ориентированных программ основаны на тестировании. Недостатком тестирования является то, что успешное его прохождение не является гарантией отсутствия ошибок в программе.

Одним из наиболее часто используемых подходов при проектировании программ со сложным поведением является автоматное программирование (программирование с явным выделением состояний). В соответствии с этим подходом поведение компонент программы со сложным поведением следует представлять в виде системы автоматов, взаимодействующих друг с другом и с неавтоматной частью программы, в состав которой включаются объекты управления. Первоначально этот подход был разработан для использования в совокупности с процедурным программированием, а затем был расширен для объектно-ориентированного программирования.

Проблемой реализации автоматных программ, недостаточно исследованной в существующих методах, является качество модификации программ при изменении требований. Для решения этой проблемы в объектно-ориентированном программировании используются рефакторинги. В настоящее время вопрос рефакторинга автоматных программ не исследован.

Основным достоинством автоматных программ является возможность повышения уровня автоматизации их верификации по сравнению с программами других классов. Кроме причины, указанной в начале автореферата, это связано также и с тем, что автоматная программа по своей структуре изоморфна структуре Крипке, используемой в *Model Checking* – одном из наиболее распространенных подходов к верификации программ.

Одной из причин, по которым верификация на основе метода *Model Checking* редко используется для повышения качества программ общего вида, является трудность автоматизации перехода от программы к модели и обратно. Для автоматных программ такая автоматизация практически осуществима, так как в этом случае программа строится по модели поведения, а при традиционном подходе – наоборот. Это утверждение также основано на том, что в автоматных программах, как и в машине Тьюринга, состояния делятся на два класса: управляющие и вычислительные. При этом управляющих состояний, в отличие от вычислительных, сравнительно немного, а верификацию в автоматных программах предлагается проводить именно для управляющих состояний.

В настоящее время существует ряд инструментальных средств, эффективно реализующих верификацию автоматных программ. При этом, однако, не известны инструментальные средства, которые позволяют *интегрировать процесс верификации с процессом разработки программ*. В частности, в известных подходах спецификация и автоматная программа существуют раздельно и обрабатываются различными инструментальными средствами.

Метод верификации на модели использует в качестве языка спецификаций один из языков темпоральной логики. Недостатком этих языков является сложность выражения требований к автоматным программам. Например, требование «Во время движения двери лифта закрыты» для автомата управления лифтом, рассматриваемого во второй главе, на языке *линейной темпоральной логики* (*LTL*) выражается следующей весьма сложной темпоральной формулой: $[(z_1 \vee z_2) \Rightarrow \overline{z_4}\mathbb{U}(z_3 \vee \overline{z_4})] \wedge \left[z_4 \Rightarrow \overline{(z_1 \vee z_2)}\mathbb{U}\left(z_5 \vee \overline{(z_1 \vee z_2)}\right)\right]$.

Вторая глава описывает разработанный подход к формализации требований к автоматным объектно-ориентированным программам. Существующий подход к формализации требований, как отмечено выше, основан на использовании темпоральных логик, наиболее удобной из которых для спецификации поведения автоматных программ является *LTL*. Однако даже для простых требований получающиеся формулы могут оказаться достаточно сложными.

Требования к объектно-ориентированным программам обычно записываются в терминах объектных интерфейсов, определяющих набор допустимых операций для объектов. Для автоматных объектно-ориентированных программ возможно использование как объектного, так и автоматного интерфейсов. Автоматный интерфейс состоит из входных и выходных воздействий, и текущего состояния. При этом независимость спецификации от текущего состояния позволяет не изменять спецификацию при замене реализации автомата.

В процессе создания автоматной программы выделяются интерфейсы автоматов. Большая часть ошибок при автоматном программировании происходит из-за неточного определения семантики элементов интерфейсов. Для решения этой проблемы можно использовать подход к формализации требований, основанный на спецификации отдельных элементов интерфейса. В объектно-ориентированном программировании для этого используется программирование по контрактам.

Традиционно контракты выражаются в форме пред- и постусловий, а также инвариантов. Применительно к состояниям автоматов эта классификация оказывается вполне естественной. Предусловиями будем считать выражения, которые должны быть истинными при входе в состояние. Постусловиями назовем выражения, которые должны быть истинными при выходе из состояния. Инварианты должны выполняться в течение всего времени, пока автомат находится в соответствующем состоянии.

Кроме контрактов состояний предлагается ввести также контракты групп состояний, входных и выходных воздействий. Примером контракта для группы состояний является инвариант «лампа включена» для группы состояний «двери лифта открыты» (объединяющей состояния «ожидание с открытыми дверьми», «двери открываются» и «двери закрываются»). Некоторые виды контрактов применимы не ко всем элементам автоматного интерфейса, что и показано в таблице.

Таблица. Совместимость элементов автоматного интерфейса и видов контрактов

	Предусловия	Постусловия	Инварианты
Состояния	+	+	+
Группы состояний	+	+	+
Входные воздействия	+	+	–
Выходные воздействия	+	–	–

Преимуществами задания требований в виде контрактов по сравнению с темпоральными формулами являются более простая форма представления и привязка к конкретному элементу автоматного интерфейса.

Проверка выполнения контрактов, как и в объектно-ориентированном программировании, может выполняться как статически, так и динамически. При этом автором предлагается *преобразовывать контракты в формулы линейной темпоральной логики*, что может быть выполнено автоматически. При статической проверке контракты используются в качестве спецификации для верификации методом *Model Checking*. Динамическая проверка контрактов основана на предложенном автором методе динамической верификации, описанном ниже. Этот метод в качестве спецификации использует формулы линейной темпоральной логики.

Динамическая верификация основана на анализе поведения программы во время выполнения. Такой анализ сводится к верификации протоколов запуска программы в различных сценариях, которые строятся в автоматных терминах. Динамическая верификация позволяет выполнить проверку работы автомата совместно с объектами управления. Такая проверка неосуществима средствами *Model Checking*, так как верификация кода объектов управления, написанного традиционно, как указано выше, трудоемка.

Известно, что любая *LTL*-формула может быть преобразована в альтернирующий автомат, причем число состояний этого автомата линейно зависит от размера формулы. Таким образом, для верификации соответствия протокола работы программы и *LTL*-спецификации достаточно по этой спецификации построить альтернирующий автомат и проверить принимается ли данный протокол полученным автоматом. Существуют три алгоритма, позволяющих выполнить такую проверку. В работе показана применимость этих алгоритмов к верификации автоматных программ и описан метод построения протоколов.

На основании предложенных методов автоматного программирования по контрактам и динамической верификации автоматных программ разработан метод формализации требований к автоматным объектно-ориентированным программам, который позволяет достичь более простой записи формализованной спецификации, и делает возможной верификацию взаимодействия автомата и объектов управления.

Третья глава описывает метод модификации автоматных программ, позволяющий уменьшить число изменений, которые могут привести к появлению ошибок. Любая программа, используемая длительное время, подвергается модификации. Это может быть связано с рядом причин: в ходе эксплуатации программы появляются новые требования, могут обнаружиться ошибки в работе программы, а также может возникнуть потребность изменить структуру программы с целью ее упрощения.

При изменении программы может нарушиться ее корректность. Таким образом, после каждой модификации необходимо обновлять спецификацию и производить верификацию программы согласно новой спецификации. Если результат верификации отрицательный, требуется проверить корректность изменений, как программы, так и спецификации. Если изменения значительные, такая проверка оказывается достаточно трудоемкой. При использовании динамической верификации даже успешная верификация не может являться гарантией корректности программы.

Одним из способов повышения надежности вносимых изменений является организация этих изменений таким образом, чтобы часть из них гарантированно не изменяла семантику программы. Такие изменения в объектно-ориентированном программировании называются *рефакторингами*. Так как существуют изменения, влияющие на поведение программы, рефакторинги не могут обеспечить корректность любых модификаций. Однако возможно организовать изменения таким образом, чтобы уменьшить размер той части изменений автомата, которая не представляется последовательностью рефакторингов. Для каждого изменения, производимого в этой части, можно предусмотреть случаи, когда изменение может нарушить корректность автомата.

Если при использовании предложенного метода корректность все же нарушена, требуется проанализировать только последнюю часть изменений, что заметно снижает трудоемкость такого анализа по сравнению с традиционным внесением изменений.

Таким образом, предлагается метод безопасного внесения изменений в автоматную программу, который состоит в разделении любого сложного изменения автомата на два этапа:

1. Рефакторинг автомата.
2. Набор модификаций, приводящих к изменению поведения автомата.

На первом этапе автомат «подготавливают» к изменениям, модифицируя его структуру, не затрагивая поведения (корректность этой фазы можно проверить автоматически, если спецификация исходного автомата была формализована). Целью выполнения этого этапа является минимизация модификаций, выполняемых на втором этапе.

Для разработки метода рефакторинга автоматных программ был проанализирован ряд работ, выполненных студентами кафедры «Компьютерные технологии» факультета «Информационные технологии и программирование» СПбГУ ИТМО (http://is.ifmo.ru/projects). В ходе этих работ студенты должны были разработать автоматные программы, управляющие тем или иным объектом. Автором диссертации

изменялось одно из требований к программе, и изучалась последовательность модификаций разработанного автомата, необходимая для отражения изменения в спецификации.

На основе результатов анализа таких изменений предлагается ввести следующие рефакторинги: группировка состояний, удаление группы состояний, слияние состояний, выделение вызываемого автомата, встраивание вызываемого автомата, переименование состояния, перемещение выходного воздействия из состояния на переход, перемещение выходного воздействия из переходов в состояние.

Для каждого из них приводится описание, мотивация (условия, при которых применение рефакторинга целесообразно), пример и техника выполнения (последовательность действий, из которых состоит рефакторинг). Также приводится доказательства корректности всех предложенных рефакторингов. Этот набор рефакторингов не является полным, но позволяет безопасно проводить ряд частых и сложных модификаций автоматных программ.

На втором этапе выполняются действия, не выражающиеся в виде последовательности рефакторингов. Назовем эти действия *базовыми изменениями*. Выделены 11 базовых изменений: добавление состояния, удаление состояния, установка признака стартового состояния, снятие признака стартового состояния, установка признака конечного состояния, снятие признака конечного состояния, добавление перехода, изменение события на переходе, изменение условия перехода, удаление перехода, перемещение перехода. Для каждого базового изменения приведен список рекомендаций, которым необходимо следовать при его выполнении.

Рассматривается пример использования метода: модификация автомата управления банкоматом в связи с изменением числа попыток ввода *PIN*-кода.

В четвертой главе описываются подходы к интеграции автоматного и объектно-ориентированного программирования. В настоящее время для реализации автоматных объектно-ориентированных программ используются графические и текстовые языки автоматного программирования. В дальнейшем в диссертации рассматриваются текстовые языки, так как работа с ними является более привычной для программистов.

Текстовые языки в основном представлены автоматными расширениями статически типизированных языков программирования (таких как *C#* и *Java*), реализованных с помощью препроцессоров – инструментов, преобразующих код на языке программирования с автоматным расширением в код на языке программирования без такого расширения. Недостатком такого подхода является сложность расширения синтаксиса языка и сложность композиции расширений.

Рассматриваются два альтернативных подхода к решению проблемы интеграции автоматного и объектно-ориентированного кода. Первый из них основан на использовании возможностей динамических языков программирования. Рассматривается разработанная автором библиотека *STROBE*, реализующая автоматное расширение динамического языка программирования *Ruby*. Эта библиотека позволяет описывать поведение классов в терминах автоматов и реализует концепцию «автоматизированные объекты управления как классы». Методы библиотеки организованы таким образом, что ее программный интерфейс выглядит как набор конструкций предметно-ориентированного языка. При этом для клиентов автоматного класса он является обычным классом системы.

Как отмечалось выше, в автоматном программировании спецификация и код программы существуют раздельно. При использовании статических методов верификации для интеграции спецификации и кода требуется, чтобы существовала возможность их статического анализа. При использовании динамических языков программирования такой анализ является трудоемким, так как семантика конструкций программы определяется только во время выполнения.

Поэтому для интеграции спецификации и кода автоматной программы целесообразно использовать метод, основанный на использовании мультиязыковых сред. В частности, в России компанией ООО «ИнтеллиДжей Лабс» разработана мультиязыковая среда *JetBrains MPS*. В этой среде программа представляется в виде абстрактного синтаксического дерева. При таком представлении программы различные расширения языков предоставляют наборы дополнительных возможных узлов абстрактного синтаксического дерева, и выбор между ними происходит явно, а не посредством неоднозначной трансляции. Узлы абстрактного синтаксического дерева являются концептами различных языков программирования. Редактирование этого дерева осуществляется с помощью редактора, который обеспечивает сходство процесса редактирования дерева с традиционным процессом редактирования текста программы. Это позволяет решить проблему комбинирования нескольких языков программирования. Использование комбинации предметно-ориентированных языков дает возможность описывать поведение программы в терминах предметной области. Для этой среды существует текстовый язык автоматного программирования *stateMachine*, однако в нем не решена проблема интеграции кода автоматной программы и ее спецификации.

Для решения этой проблемы автором было разработано расширение языка *stateMachine*, названное *stateSpec*. Это расширение позволяет задавать спецификацию программы в виде формул линейной темпоральной логики и контрактов. При этом

конструкции, описывающие спецификацию, отображаются и редактируются как часть описания автоматной программы. Ниже приведен пример совмещения кода и спецификации автомата управления кофеваркой:

```
initial state(ReadyToUse) {
  on unPlugged do {<no statements>} transit to (TurnedOff)

  invariant {
    voltageTreshold == 220
    currentVoltage < voltageTreshold
  }

  require for ReadyToUse
    isTurnedOn
  }
}
specification {
  G F !(Broken)
  ! ReadyToUse R TurnedOff
  currentVoltage < voltageTreshold U Broken
}
```

При редактировании происходит проверка синтаксической корректности спецификации в реальном времени. Также для облегчения ввода спецификации реализована возможность выбора и вставки конструкций языка линейной темпоральной логики и имен элементов автоматного интерфейса соответствующего автомата.

Разработанное языковое расширение позволяет запустить процесс верификации программы из среды *MPS*. При запуске этого процесса описание автоматной программы и спецификация **автоматически** преобразуются во входной формат верификатора, в качестве которого выбран *NuSMV*. При этом контракты автоматически преобразуются в формулы темпоральной логики. После преобразования производится запуск верификатора и выдается сообщение о соответствии или несоответствии программы спецификации. В случае несоответствия производится анализ контрпримера – он преобразуется в путь в автоматной программе.

Разработанное расширение решает проблему интеграции спецификации и кода автоматной программы, делая возможным их синхронное изменение и мгновенную проверку соответствия программы спецификации.

Пятая глава содержит описание результатов внедрения предложенных методов и расширения предметно-ориентированного языка.

Метод интеграции кода и спецификации автоматной программы, а также предметно-ориентированный язык *stateSpec*, описанные в данной работе, использовались:

- в компании *ООО «ИнтеллиДжей Лабс»* при разработке системы учета дефектов *YouTrack*;
- в учебном процессе СПбГУ ИТМО по курсу «Применение автоматов в программировании» кафедры «Компьютерные технологии».

Опишем внедрение более подробно. В компании *ООО «ИнтеллиДжей Лабс»*, работающей на мировом рынке под торговой маркой *JetBrains*, разработана система учета дефектов *YouTrack*. Эта система реализована на базе среды мультиязыкового программирования *MPS*. Система представляет собой клиент-серверное интернет-приложение, в котором клиентская часть выполняется в интернет-обозревателе пользователя. Для уменьшения времени отклика в системе используется технология *AJAX* (*Asynchronous Javascript And Xml*).

Часть поведения системы, связанная с построением реактивных компонентов пользовательского интерфейса, реализована в виде автоматов на диалекте языка *stateMachine*, в котором в качестве базового используется язык *JavaScript* вместо языка *Java*. Эти автоматы получают на вход информацию о действиях пользователя и ответы, полученные от сервера. В качестве выходных воздействий используются команды обновления пользовательского интерфейса и выполнения запросов к серверу. Примерами автоматов являются автомат управления списком дефектов на экране и автомат управления интерфейсом строки поиска.

Для системы *YouTrack* была создана специальная версия расширения языка *stateSpec* для обеспечения работы с языком *JavaScipt*. Успешная реализация такой модификации свидетельствует о хорошей переносимости разработанного расширения. С использованием модифицированного расширения была описана спецификация перечисленных выше автоматов и обеспечена возможность их верификации в процессе разработки.

Заключение

В диссертации получены следующие научные результаты:

1. Создан метод формализации требований к автоматным объектно-ориентированным программам на основе использования формул темпоральной логики и автоматного программирования по контрактам, а также статической и динамической верификации.
2. Разработан метод интеграции кода и формализованных требований к автоматным объектно-ориентированным программам. Разработан предметно-ориентированный язык, реализующий созданный метод.

3. Создан метод внесения изменений в автоматные программы на основе рефакторинга, уменьшающий число изменений, которые могут привести к появлению ошибок.
4. Составлен каталог рефакторингов автоматных программ. Представлено доказательство эквивалентности программы до и после рефакторинга.
5. Полученные результаты внедрены в практику разработки программного продукта *YouTrack* в компании *ООО «ИнтеллиДжей Лабс»* и учебный процесс в СПбГУ ИТМО.

Список публикаций

1. Степанов О. Г., Шалыто А. А., Шопырин Д. Г. Предметно-ориентированный язык автоматного программирования на базе динамического языка RUBY // Информационно-управляющие системы. 2007. № 4, с. 22–27.
2. Степанов О. Г. Метод автоматической динамической верификации автоматных программ // Научно-технический вестник СПбГУ ИТМО. № 53, с. 221–229.
3. Stepanov O., Shalyto A. A. Method for Automatic Runtime Verification of Automata-Based Programs // Proceedings of Spring/Summer Young Researchers' Colloquium on Software Engineering 2008, Vol. 2, pp. 19–23.
4. Борисенко А., Федотов П., Степанов О., Шалыто А. Разработка надежного программного обеспечения со сложным поведением // Сборник трудов конференции 5th Central and Eastern European Software Engineering Conference in Russia, с. 125–128.

Тиражирование и брошюровка выполнены в
центре «Университетские телекоммуникации»
Санкт-Петербург, Саблинская ул. 14; тел: (812)233-4669
Тираж 100 экз.