УДК 681.3.06

ПРОБЛЕМЫ РЕИНЖИНИРИНГА ПРОГРАММНЫХ LEGACY-СИСТЕМ

Н.Д. Пашковец, О.В. Бабак

Міжнародний науково-навчальний центр інформаційних технологій та систем НАН України та МОН України,
03680, Київ, проспект Академіка Глушкова, 40, корпус 7, (044) 526 4187,
dep175@irtc.org.ua

В работе изложен подход к анализу программных систем с целью автоматизации разбора и отображения логики, лежащей в основе их функционирования. Главное внимание направлено на выделение бизнес-объектов и идентификации бизнес-терминов как элементарных составляющих, на которых базируется описание и построение логических схем и структур программных legacy-систем. Изложено обоснование предлагаемого подхода к реинжинирингу программных legacy-систем и определяются условия корректного построения комплекса бизнес-правил. Предложена методика и алгоритмы автоматизации определения паттернов в модулях legacy-систем, выделение которых позволяет решать проблемы обобщения, унификации и классификации создаваемых бизнес-правил в рамках конкретной legacy-системы или ряда legacy-систем, относящихся к одному типу, при проведении их реинжиниринга. Разработана общая технология построения БП в виде взаимосвязанных средств автоматизации построения БП (САПБП) для выполнения реинжиниринга программных legacy-систем.

The article describes the approach to the analysis of software systems to automate the parsing and display logic that underlies their functioning. The main attention is directed to the provision of business objects and identification of business terms as the elementary constituents, which is based on the description and the construction of logic circuits and structures of software legacy-systems. Set out the rationale of the proposed approach to reengineering legacy-software systems and defines the conditions for constructing a correct set of business rules. The technique and algorithms for automation of certain patterns in the modules of legacy-systems, the allocation that solves the problems of consolidation, harmonization of classification and create business rules within a particular legacy-system or some legacy-systems belonging to the same type, during their re-engineering. A general technique for constructing BP in the form of interconnected building automation BP (SAPBP) for performing software reengineering legacy-systems.

Введение

Современное стремительное развитие ИТ-технологии часто приводит к быстрому устареванию программных систем (legacy-систем), что порождает проблему их модификации (реинжиниринга**)** в соответствии с требованиями современных технологий. Решать указанную проблему можно двумя путями, а именно:

– отказаться от старой программной системы и разрабатывать новую;

– выполнять реинжиниринг старой системы.

В первом случае придется разрабатывать заново всю бизнес-логику программной системы, которая, а это очень существенно, постоянно совершенствовалась и менялась в процессе многолетней эксплуатации старой системы, а, следовательно, не всегда существует документация по текущей ее версии. Чтобы вновь разрабатываемая система по зрелости бизнес-логики достигла уровня старой, может потребоваться не один год. Поэтому, при решении указанной проблемы все чаще отдают предпочтение реинжинирингу.

Отличия между старой и новой системами определяются различием архитектур программного обеспечения и моделей данных. Общим для обеих систем является бизнес-логика, которая не меняется при преобразовании одной системы в другую, т. е. она инвариантна относительно такого преобразования (это следует из условий реинжиниринга). Следовательно, бизнес-логика переходит без изменений в новую систему и инкапсулируется в отдельные классы.

В связи с тем, что в мире накоплено значительное количество legacy-систем, и с течением времени все большая их часть переходит границу морального старения, актуальность процесса реинжиниринга с каждым годом возрастает. Поэтому возникает необходимость разработки методов и алгоритмов автоматизации реинжиниринга программных систем, для быстрого и эффективно выполнения их модификации.

Специалист по реинжинирингу работает с двумя моделями: со старой, которую он исследует, и с новой, которую он формирует. Предлагаемая методология позволяет использовать для этих работ такие средства как Rational Rose, наличие в котором механизма спирального моделирования предоставляет возможность после анализа каждого бизнес-процесса вносить дополнения и изменения в диаграммы новой модели. При этом шаг за шагом, итерация за итерацией, старая модель системы постепенно преобразуется в новую, которая затем используется для генерации и/или написания программного кода новой системы. За отправную точку при рассмотрении этого вопроса принят тот факт, что в любом программном продукте можно определить обобщенную модель обработки информации, которая связана с реальными объектами и понятиями предметной области, называемыми ее бизнес-объектами.

Бизнес-объекты – базовые понятия и определения конкретной предметной области, участвующие в решении исходной задачи. Всякое преобразование или переименование логической величины в программе в процессе решения задачи и передача ее между программными модулями или проектами не меняет сути бизнес-объекта. Допускаются порождения новых бизнес-объектов по некоторым правилам на основе базовых логических величин. Такие бизнес-объекты будем называть порождаемыми. При передаче порождаемого бизнес-объекта из одного программного модуля в другой он приобретает статус базового бизнес-объекта, но в исходном программном модуле с ним будет связано правило, описывающее условия порождения этого бизнес-объекта, т. е. бизнес-правило порождения бизнес-объекта или просто **бизнес-правило порождения**. Все базовые бизнес-объекты являются входными данными, а выходными данными могут быть как базовые, так и порождаемые бизнес-объекты.

Бизнес-термины – наименования бизнес-объектов, которые определяются для всего программного проекта в целом.

Условия взаимодействия бизнес-объектов определяют общий **бизнес-процесс** [1] обработки информации, который является каркасом программного обеспечения системы. Поэтому первым действием реинжиниринга будет освобождение каркаса legacy-системы от надстройки, т. е. на основе анализа исходного кода программ аналитики выделяют бизнес-конструкцию, отделяя ее от технической программной надстройки, связанной с особенностями реализации процесса, а также от его сервисной надстройки (если она устарела).

После разбора функциональной структуры legacy-системы и определения ее бизнес-объектов аналитик на первой стадии реинжиниринга (Preprocessing) строит схемы информационных потоков и выделяет множество действий, производимых над бизнес-объектами. Эти действия будем называть **бизнес-правилами** (БП) [2, 3]. На второй стадии реинжиниринга определяются условия и порядок выполнения БП, что является формализованным содержанием **бизнес-процессов**. Совокупность допустимых условий, при которых возможно выполнение БП, будем называть **событиями** этого БП. На третьей стадии реинжиниринга аналитики осуществляют классификацию и обобщение элементов множества БП, с учетом всех их возможных событий с целью оптимизации и минимизации количества БП.

БП описываются на структурном языке, типа ENGLISH STRUCTURE, что обеспечивает отображение полученного бизнес-процесса в среде современных технологий, а также с минимальными затратами позволяет выполнить реализацию новой технической и сервисной надстройки.

Постановка задачи. В работе поставлена задача построения конкретных методов анализа и разбора программных legacy-систем в целях их реинжиниринга на основе разработанной концепции определения и описания БП.

Решение задач реинжиниринга legacy-систем. К области реинжиниринга legacy-систем можно отнести следующие проблемные задачи, которые возникают перед пользователями старых информационных технологий:

конвертирование программных комплексов с одной системы программирования в другую (например, конвертирование программного комплекса на языке Assembler OS\390, в язык COBOL-II OS/390, программной системы на языке PL/1 AS/400, в систему COBOL-II AS/400 для Израильской фирмы “MENORA” в 2005 году;

сопряжение по управлению новых программных комплексов с legacy-системами – путем трансформации языка управления заданиями, например, JCL для OS/360/370/390 z/OS или CL для AS/400, в клиент-серверную технологию;

сопряжение по данным legacy-систем с новыми программными комплексами, при котором старые СУБД заменялись современными, путем замены запросов к СУБД IMS/2, в языке COBOL, на запросы к СУБД ORACLE в PL/SQL и замены программного комплекса, работающего с системой EASYTRIEVE PLUS STANDARTS на аналогичную COBOL-систему;

сопровождение legacy-систем, не достигшим границ морального старения (внесение изменений и дополнений в процессе ее эксплуатации). Реализация выполнялась для авиакомпании DELTA AIRLINES (USA) в 1997–2000 году в среде IBM OS/390 на языках Assembler, COBOL, PL/1, C, SAS, NATURAL, IDEAL, RPG, FORTRAN, JCL, EXEC, REXX, XEDIT и др.;

отображение логики legacy-систем, которые уже перешли границы морального устарения, во множество БП. В этом направлении для авиакомпании DELTA AIRLINES (USA) в 2001 году выполнялись описания логики устаревших программных систем, написанных на языках программирования ASSEMBLER, COBOL, PL/1, C, NATURAL, JCL и др.

Основные концепции построения БП. Различие архитектур и моделей старой и новой систем приводит к различию процедур, управляющих передачей бизнес-объектов, но бизнес-логика при этом остается неизменной [4]. БП сами по себе, вне контекста всего бизнес-процесса и его управляющей составляющей, никакого интереса не представляют, так как непонятно как их использовать и где их место в новой модели системы. Поэтому предметом рассмотрения реинжиниринга должен быть бизнес-процесс в целом, от старта и до его завершения. При этом монолитные старые программные системы в процессе реинжиниринга разделяются (исходя из функциональности) на отдельные бизнес-компоненты, из которых строятся новые бизнес-процессы в рамках инвариантной бизнес-логики. Учитывая что главная задача аналитика – определение тех участков алгоритма, которые содержат БП, т. е. необходимо разделить весь вычислительный процесс на его управляющую составляющую и бизнес-логику. Управляющая часть процесса требуется для понимания путей передачи данных, которые в новой архитектуре будут представлены по-другому, а именно – через взаимодействие объектов. Бизнес-логика, как было отмечено, переходит без изменений и инкапсулируется в контейнеры.

Написание БП, включающего все события какого-то бизнес-процесса, делает его абсолютно необозримым и который с трудом поддается осмыслению. Собирание в одном БП всех условий выполнения событий от начальной точки анализируемого бизнес-процесса, до его конечной точки следуя по какой-то ветке программы опишет только цепочку событий, которая приводит к активизации данного события. Такое БП не отразит других цепочек, которые также приводят к активизации этого же события. Цепочки следования по другим веткам программы будут отражены в таких БП, у которых (вероятнее всего) могут быть повторяющиеся (дублирующиеся) части. Это в значительной степени будет загромождать формируемую модель решаемой задачи. Несмотря на то, что указанные БП описывают активизацию одного и того же события, не будет отражена динамическая связь между этими правилами. Значит, в этом случае преобразовать архитектуру системы (например, под методологию объектно-ориентированного программирования) с сохранением требуемой бизнес-логики, будет довольно трудно.

Избежать перечисленных недостатков можно, если при написании БП ориентироваться на их дальнейшую обработку с применением технологии “автоматного” программирования. То есть в конечном итоге БП должны приобрести такую форму, чтобы генерация выходного кода осуществлялась с помощью конечного автомата в виде диаграмм состояний. Диаграммы состояний (как основа автоматного подхода к реализации различных задач, решаемых программным путем), стали неотъемлемой частью основных принципов объектно-ориентированного программирования. Кроме того, такие диаграммы входят в стандарт UML и вписываются в методологию SWITCH-технологии.

Формирование матрицы активизации БП. В процессе определения аналитиком конкретного БП, как элемента комплекса БП, создается набор логических условий, которые вызывают активизацию данного БП. Опыт добывания БП из программного унаследованного кода показывает, что из множества условий активизации БП, имеется хотя бы одно, выполняющее начальное включение БП [5]. Такие логические условия в комплексе БП назовем **первичными.**

Первичное логическое условие может относиться к одному или нескольким БП. При первой активизации БП условие адекватно первичному, при последующих – выполняет роль **дополнительных** условий. Все множество формируемых условий комплекса БП представляется в виде матрицы размерностью **(m+n, m)**, где **m** – количество БП, **m+n** – количество используемых в них условий, а **n** – количество первичных и дополнительных условий. Подматрица **(m, m)** предназначена для фиксации и отслеживания использования первичных и дополнительных условий. В подматрице **(n, m)** фиксируется и отслеживается использование только дополнительных условий.

Принцип матричного отображения множества условий на множество БП используется при построении статических и динамических векторов состояния [6], где матрица – набор условий запуска БП, а столбцы соответствуют векторам условий активизации соответствующих БП. Рабочая матрица условий запуска БП (далее – матрица запуска БП) может быть построена автоматически путем просмотра всех БП и выборки из них логических переменных и условий запуска БП.

Первоначально матрица запуска БП формируется вручную и расширяется по мере добавления БП и условий. Таким образом предоставляется возможность следить за формированием условий активизации создаваемых БП и соблюдением их уникальности. Этот процесс поддается автоматизации с выполнением некоторой оптимизации относительно вводимых логических переменных в процессе написания БП, а также для проверки непротиворечивости запуска БП, что значительно повысит производительность и качество БП.

Для формирования матрицы условий запуска БП использован следующий формат объявления логических переменных:

F(I):<имя флажка> – для первичных флажков, где буква **F** – признак первичности флажка, а индекс **I** – номер БП (имеет диапазон изменения от **1** до **m**);.

S(J): <имя флажка> – для дополнительных флажков, где буква **S** – признак дополнительного (вторичного) условия, а индекс **J** имеет диапазон изменения от **m+1** до **n**.

После построения матрицы запуска БП создается статический вектор состояния флажков размерностью **m+n**, в котором фиксируется текущее состояние всех логических переменных комплекса БП. Если комплекс БП построен корректно, то для всякой конфигурации вектора состояния найдется один и только один адекватный ему столбец в матрице запуска БП.

Корректность построения комплекса БП. Будем считать, что комплекс БП построен корректно, если входящие в его состав БП удовлетворяют условиям: непротиворечивость, необходимость, достаточность и полнота.

БП являются непротиворечивыми, если все столбцы матрицы запуска БП уникальны.

БП являются достаточными, если для всякой текущей конфигурации вектора состояния имеется адекватный столбец в матрице запуска БП.

Если столбец в матрице запуска БП активизировался хотя бы один раз в процессе работы системы, то БП является необходимым.

Свойство полноты БП в данном случае вытекает из выполнения условий необходимости и достаточности.

Для унификации активизации БП различных видов можно ввести отдельный вектор активизации правил, представляющий собой бинарный вектор, размер которого равен количеству БП. В каждый момент времени в векторе активизации “включен” только один элемент, соответствующий индексу активного БП. В конце отработки БП этот элемент “выключается”, а включается элемент, соответствующий БП активизируемому следующим. Таким образом, вектор активизации отражает порядок и время выполнения БП, что позволяет организовать обработку очередей для параллельных процессов и реализацию мультизадачного режима.

Операции, выполняемые на стадии Preprocessing. При разборе функциональной структуры программной legacy-системы выполняются такие работы:

1) построение дерева вызовов модулей системы из активной части системы, установления порядка и условий вызовов модулей (неработающая (мертвая)) часть системы не учитывается.

2) определение межмодульных связей для построения схемы внутрисистемных и межсистемных информационных потоков с занесением входных и выходных данных каждого модуля в специальную таблицу переменных модуля. Отмеченные внешние данные образуют межмодульные связи для анализируемого модуля в системе. Выполняется настройка парсера определения межмодульных связей на распознавание тех синтаксических конструкций, которые реализуют прямую или косвенную передачу данных между модулями (операторы вызова программ, общие области данных, области связи и т. д.) с целью определения имен передаваемых данных и последующей их идентификацией в таблице переменных модуля. На основании этой информации строится таблица межмодульных связей, в которой отображается передача данных между модулями, рассматриваемые как предполагаемые кандидаты в бизнес-объекты.

3) отслеживание связей наследования между внутри модульными переменными заключается в построении цепочек наследования переменных, связанных между собой общностью пересылаемых данных. Эти цепочки идентифицируются именами, которые рассматриваются как кандидаты в бизнес-термины. Идентификация цепочек наследования переменных выполняется по следующим правилам:

проверяется первый и последний элементы цепочки наследования на предмет принадлежности их к входным и выходным данным соответственно. Проверка выполняется на основе таблицы переменных модуля;

если в цепочке наследования присутствует входное данное, то кандидату в бизнес-объект, соответствующему этой цепочке наследования, будет присвоено имя входной переменной;

если в цепочке наследования отсутствует входное данное, но присутствует выходное данное, то кандидату в бизнес-объект, соответствующему этой цепочке наследования, будет присвоено имя выходной переменной;

если в цепочке наследования отсутствуют входное и выходное данное, то кандидату в бизнес-объект, соответствующему этой цепочке наследования, будет присвоено имя первой переменной в цепочке, если она порождена с участием каких-либо входных данных. В противном случае предполагается, что цепочка не связана с бизнес-объектами и исключается из рассмотрения.

Цепочка наследования может состоять из одной переменной, если в модуле не пересылает данные другим переменным, но участвует в порождении другой переменной, либо быть пассивной.

Присвоенные имена кандидатов в бизнес-термины с выделенными цепочками наследования заносятся в таблицу имен наследования модуля.

Таблица предполагаемых бизнес-терминов каждого модуля прилагается к соответствующему модулю при передаче его аналитику для координации проводимого анализа и определения списка бизнес-терминов.

Рассмотренные методы и механизмы автоматического построения БП применимы для тех частей legacy-систем, которые оперируют базами данных, а также для организации последовательности действий, выполняемых в ходе вычислительного процесса, для адекватного отображения этих процессов в новой системе (обработка SQL-вложений и построение БП для JCL-заданий) [6]. Правильное решение этих задач обеспечивает не только эффективный реинжиниринг, но и квалифицированное современное сопровождение legacy-систем, с учетом особенностей их межсистемных стыковок современными программными системами как по обмену данными, так и по управляющим воздействиям.

Выявления паттернов в анализируемой legacy-системе. На втором этапе выполнения реинжиниринга legacy-систем решается задача выделения и поиска паттернов для учета их при написании БП [7], их обобщения и унификации создаваемых БП, а также их классификации в рамках конкретной legacy-системы, или для ряда идентичных legacy-систем. На основе выделенных паттернов строится специальное структурированное хранилище БП (репозиторий) и инструментальный интерфейс для работы с ним как в автоматизированном, так и в интерактивном режимах.

В процессе анализа программной системы аналитик, по возможности, выделяет “подобные” программные конструкции или блоки (паттерны), содержащие некоторое множество операторов исходной программы и имеющие одинаковую структуру с точки зрения логики выполняемых действий. При наличии общей структуры эти множества операторов отличаются используемыми именами переменных или подмножеством операторов, выполняющих не основные операции. Следовательно, паттерны формально можно представить тройкой < **B, P, V** >, где **B** – общая (базовая) логическая структура паттерна, $\mathbf{P} = \{p_i\}$ – множество используемых имен в базовой структуре, $\mathbf{V} = \{ v_j \}$ – подмножество операторов для не основных операций.

В задачу аналитика входит определение части **B** для построения поискового образа, по которому будет выполняться поиск паттернов в программных модулях анализируемой системы. Оптимальное определения **B** состоит в варьировании границы и конфигурации его элементов из подмножества **V** и изменением состава множества **P**. Сравнение результатов поиска при варьировании **B** приводит к определению оптимальной структуры **B**, которая при поиске выявила максимальное количество паттернов. Хотя части **P** и **V** не участвуют в поисковом процессе, но при формальном описании поискового образа должны быть учтены. Для p_i и v_j необходимо указать: место их расположения в **B**, максимальное количество линий и максимальное количество символов в линиях. Часть **B** всегда должна присутствовать в поисковом образе, в то время как присутствие частей **P** и **V** не является обязательным.

Эффективная работа с паттернами зависит от наличия инструментария, выполняющего следующие работы:

- настройка на грамматику языка программирования анализируемого программного модуля для проведения синтаксического анализа текста по построенным на стадии Preprocessing таблицам лексем программного модуля и выделения синтаксических элементов, осуществляя соответствующую разметку;
- построение системы иерархических закладок, представляющую собой специальную структуру поиска участков кода (паттернов) по задаваемым условиям и правилам, относящимся к переменным, которые являются кандидатами в бизнес-термины. Это осуществляется сканированием выполняемых операций на **n**-м количестве шагов в прямом и обратном направлениях, группирование по поисковым объектам выполняемых операций и отслеживание бизнес-объектов внутри модулей;
- формализованное описание поисковых образов паттернов и занесение их в набор данных паттернов анализируемой системы репозитория, а также настройка синтаксического анализатора разметок элементов программных модулей на формализованное описание паттернов. Это предоставляет возможность аналитику проверить целесообразность выделенного паттерна, исходя из количества найденных в системе подобных паттернов (путем разметки или группирования). Если число подобных паттернов невелико, то даже небольшая корректировка в описании паттерна, может привести к значительному увеличению их количества. Из построенных паттернов выбираются с совпадающими или альтернативными операциями, что сокращает число создаваемых правил или дублирующих частей системы;
- отметка в исходном программном модуле границ, которые соответствуют уже сформированным и занесенным в репозиторий БП. Это служит источником разметки анализируемого модуля по БП и анализа модуля на полноту его логического покрытия построенными БП, а также контроль взаимно однозначного соответствия между модулями анализируемой системы и БП в репозитории.

Общая технология построения БП. Описанные принципы и методы реинжиниринга legacy-систем положены в основу технологии построения БП с помощью средств автоматизации построения БП (САПБП) для промышленного реинжиниринга разных программных legacy-систем [8]. САПБП – это интерактивная система, в состав которой входит ряд программных средств, предназначенных для автоматизированной обработки отдельных шагов технологического процесса реинжиниринга, в сочетании с методическими и методологическими правилами и указаниями. Legacy-системы рассматриваются как единый комплекс программных модулей с учетом потоков данных, посредством которых осуществляется обмен бизнес-объектами между компонентами модифицируемого программного комплекса. Передача бизнес-объектов между модулями системы осуществляется через внешние файлы, как главными составляющими компонентами при отображении проблемной области в новое программное представление.

Определим три положения передачи данных через файлы между модулями, которые настолько очевидны, что не требуют математического доказательства.

1. О передаче данных файла F1 от модуля M1 к модулю M2 можно говорить только в том случае, если хронологически модуль M2 выполняется после модуля M1, и при этом файл F1 в модуле M1 является выходным или обновляемым, а в модуле M2 – входным.

2. Модули M1 и M2 должны одновременно принадлежать одному из маршрутов дерева вызова программ, т .е. на дереве вызова должен существовать маршрут выполнения программ, который включает в себя оба модуля M1 и M2.

3. Пересечение множества **A** выводимых данных в файл F1 в модуле M1 и множества **B** вводимых данных из файла F1 в модуле M2, не должно быть пустым:

$$A \cap B \neq \varnothing.$$

Именно проверку этих условий необходимо выполнить для окончательного установления факта передачи данных через файлы между двумя модулями.

Выводы. При разработке основных методических концепций и средств автоматизации добывания БП при реинжиниринге legacy-систем получены следующие результаты:

–для упорядочения и регламентации работы с логическими переменными, используемые в БП в качестве выражения различных условий, проведена их классификация;

–выполнена классификация и предложен механизм отслеживания состояния логических переменных для организации последовательности вызова БП. Определены правила и методы построения статических и динамических векторов состояния для текущего отслеживания условий выполнения БП и предложена методика построения матрицы условий запуска БП, которая используется с одной стороны – как карта контроля при написании БП, а с другой – как механизм определения момента активизации БП. Определены условия корректного построения комплекса БП и принята методика их проверки;

–разработан механизм отслеживания наследования бизнес-объектов для всей legacy-системы в целом;

–определены методики и алгоритмы выявления паттернов в модулях legacy-систем, проведено обобщение и унификация создаваемых БП, а также их классификации в рамках анализируемой legacy-системы;

–предложены методики построения специальных структурированных хранилищ БП (репозитории), а также инструментального интерфейса для работы с репозиториями как в автоматизированном, так и в интерактивном режимах;

–разработана общая технология построения БП в виде взаимосвязанных средств автоматизации построения БП (САПБП) для выполнения реинжиниринга программных legacy-систем;

–определены принципы настройки применяемых в САПБП парсеров на грамматики языков программирования модулей legacy-систем, что позволяет расширить диапазон применения САПБП, провести стандартизацию и унификацию применения средств автоматизации для использования их в различных проблемных областях.

1. *Tom Debevoise*. Business Process Management with a Business Rule Approach.(Business Knowledge Architects, 2005) .
2. *Barbara* & GOLDBERG, Larry (2006 October 9). The Business Rule Revolution. Happy About.
3. *Principles* of Business Rule Approach, Ronald G. Ross (Aw Professional, 2003) .
4. *Анисимов А.В., Белодед В.В. , Пашковец Н.Д., Бабак О.В.* Реализация реинжиниринга программных legacy-систем // УСиМ. – № 6. – 2008 – С. 40–48.
5. *Гриценко В.И., Анисимов А.В., Пашковец Н.Д., Бабак О.В.* Автоматизация построения бизнес-правил в процессе реинжиниринга программных legacy-систем на этапе анализа их функциональных структур. // УСиМ. – № 3. – 2009 – С. 56–64.
6. Гриценко В.И., Анисимов А.В., Пашковец Н.Д., Бабак О.В. Построение бизнес-правил для SQL-вложений и JCL-заданий. // УСиМ. – № 4 – 2009– С. 58–66.
7. Гриценко В.И., Анисимов А.В., Пашковец Н.Д., Бабак О.В. Анализ программных legacy-систем на предмет определения объектов – кандидатов в бизнес-термины. // УСиМ. – № 5. – 2009– С.69–75.
8. Гриценко В.И., Анисимов А.В., Пашковец Н.Д., Бабак О.В. Общая технология построения бизнес-правил при реинжиниринге программых систем. // УСиМ. – № 6. – 2009.– С. 3–11.