УДК 681.3.06

В.И. Гриценко, А.В. Анисимов, Н.Д. Пашковец, О.В. Бабак

Анализ программных *legacy*-систем для определения объектов – кандидатов в бизнес-термины

Статья продолжает серию публикаций по вопросам реинжиниринга программных *legacy*-систем. Изложен подход к разработке методики и алгоритмов 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 *legacy*-систем, относящихся к одному типу, при проведении их реинжиниринга.

The present article continues a series of publications on the problems of reengineering of the program legacy-systems. An approach to the development of the methods and algorithms of automating the definition of patterns in the legacy-systems modules is stated. Their selection makes it possible to solve the problems of the generalization, unification and classification of the created business-rules in the framework of a concrete legacy-systems or legacy-systems belonging to the one type under their reengineering.

Стаття продовжує серію публікацій з питань реінжиніринга програмних *legacy*-систем. Описано підхід до розробки методики та алгоритмів автоматизації визначення паттернів у модулях *legacy*-систем, виділення яких дозволяє розв'язати проблеми узагальнення, уніфікації та класифікації створених бізнес-правил в межах конкретної *legacy*-системи або *legacy*-систем, які належать до одного типу, при виконанні реінжиніринга програмних *legacy*-систем.

Введение. Статья продолжает серию публикаций [1–3], где изложен комплексный подход к реинжинирингу программных *legacy*-систем с использованием бизнес-правил (БП) как основополагающего логического понятия [1], на котором базируется предлагаемый подход. В публикациях [1, 2] определены три основных этапа реинжиниринга программных *legacy*-систем, приведено обоснование предлагаемого подхода, описаны некоторые алгоритмы, ориентированные на автоматизацию построения БП для первого этапа. В [3] изложен подход к построению БП при выполнении реинжиниринга *SQL*-обращений, встроенных в программы *legacy*-систем, и описаны алгоритмы построения БП для *JCL*-заданий, инициирующих порядок выполнение программ *legacy*-систем.

В настоящей статье основное внимание акцентировано на описании алгоритма определения паттернов в модулях *legacy*-систем. Выделение паттернов способствует решению проблемы обобщения и унификации создаваемых БП, а также их классификации в рамках конкретной *legacy*-системы или для ряда *legacy*-систем, подобных по типу. Это позволяет построить специальные структурированные хранилища БП (репозитории) и разработать инструментальный интерфейс для работы с ними как в автоматизированном, так и в интерактивном режимах.

Аналогичные режимы работы предусмотрены при проведении анализа модулей *legacy*-систем для выделения отдельных, логически завершённых, программных блоков. Предложена также методика по выделению указанных блоков программ.

Постановка задачи

Разрабатываются конкретные методы и механизмы анализа модулей программной *legacy*-системы с целью определения объектов – кандидатов в бизнес-термины для программной *legacy*-системы. Для этого необходимо наличие инструментария, способного выполнять следующие работы:

- настройку на грамматику языка программирования анализируемого программного модуля, т.е. требуется настраиваемый на грамматику синтаксический анализатор, который, используя построенные на стадии *Preprocessing* таблицы лексем программного модуля, выполняет выделение синтаксических элементов и осуществляет соответствующую разметку;

• построение системы иерархических закладок, представляющую собой специальную структуру поиска участков кода (паттернов) по задаваемым условиям и правилам, относящимся к переменным – кандидатам в бизнес-термины; это позволит осуществлять сканирование по выполняемым операциям на *n*-е количество шагов в прямом и обратном направлениях, а также производить группирование по поисковым объектам выполняемых операций и прослеживать продвижения бизнес-объектов внутри модулей;

• формализованное описание поисковых образов паттернов и занесение этих описаний в набор данных паттернов анализируемой системы в репозитории, а также настройку синтаксического анализатора разметки элементов программных модулей на формализованное описание паттернов; данный инструментарий предоставляет возможность 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 количества найденных прототипов; следовательно, аналитик получает возможность выбора паттерна с наибольшим числом совпадающих или альтернативных операций (с точностью до использования в них переменных) для сокращения количества создаваемых правил или поиска дублирующих частей системы;

• отметить в исходном программном модуле границы, соответствующие уже сформированным БП и занесенным в репозиторий. Это позволит выполнять разметку анализируемого модуля в соответствии с формируемыми БП и проанализировать модуль с точки зрения полноты его логического покрытия построенными БП. При этом устанавливается и контролируется взаимно однозначное соответствие между модулями анализируемой системы и БП в репозитории.

Описание алгоритма выявления паттернов в анализируемой *legacy*-системе

В процессе анализа программной системы аналитик, по возможности, выделяет «подобные» программные конструкции или блоки, под которыми подразумеваются некоторые множества операторов исходной программы, имеющие, с учетом логики выполняемых действий, одинаковую структуру. При наличии общей структуры эти множества операторов отличаются используемыми именами переменных или подмножеством операторов, выполняющих второстепенные операции. Следовательно, паттерны формально можно представить тройкой $< B, P, V >$, где

B – общая (базовая) логическая структура паттерна;

$P = \{ p_i \}$ – множество используемых имен в базовой структуре;

$V = \{ v_j \}$ – подмножество операторов для второстепенных операций.

Аналитик должен определить часть B с оптимальной точностью для построения поискового образа, по которому будет выполняться поиск паттернов в программных модулях анализируемой системы. Для оптимального определения B аналитик может варьировать границы и конфигурацию его элементов, используя подмножество V, и даже меняя состав множества P. Путем сравнения результатов поиска при варьировании B можно определить оптимальную структуру B, которая при поиске выявила максимальное количество паттернов. Части P и V, хотя не участвуют в поисковом процессе, 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 наблюдаться в поисковом образе, в то время как для частей P и V это не является обязательным.

Паттерны и определение их поискового образа показаны во фрагменте программы на языке *COBOL* в примере 1:

Пример 1.

```
1.  QKSD001F-GET.
2.    READ QKSD001F-FILE INTO PASS-RECORD
3.    IF  QKSD001F-STATUS = '00'
4.      SUBTRACT PASS-RESULT FROM PASS-RESULT
5.    ELSE
6.      IF  QKSD001F-STATUS = '10'
7.        ADD 16 TO ZERO GIVING PASS-RESULT
8.      ELSE
9.        ADD 12 TO ZERO GIVING PASS-RESULT
10.         DISPLAY '* CBLCBLC2 QKSD001F-
            Failed-READ-attempt...'
11.           UPON CONSOLE
12.         MOVE QKSD001F-STATUS TO IO-STATUS
13.         PERFORM DISPLAY-IO-STATUS
14.       END-IF
15.     END-IF
16.     EXIT.

17.  QKSD001F-OPEN.
18.    OPEN INPUT QKSD001F-FILE
19.    IF  QKSD001F-STATUS = '00'
20.      SUBTRACT PASS-RESULT FROM PASS-
         RESULT
21.    ELSE
22.      ADD 12 TO ZERO GIVING PASS-RESULT
23.      DISPLAY '* CBLCBLC2 QKSD001F-Failed-
         OPEN-attempt...'
24.        UPON CONSOLE
25.      MOVE QKSD001F-STATUS TO IO-STATUS
26.      PERFORM DISPLAY-IO-STATUS
27.    END-IF
28.    EXIT.

29.  QKSD001F-CLOSE.
30.    CLOSE QKSD001F-FILE
31.    IF  QKSD001F-STATUS = '00'
32.        SUBTRACT PASS-RESULT FROM PASS-
           RESULT
33.    ELSE
34.      ADD 12 TO ZERO GIVING PASS-RESULT
35.      DISPLAY '* CBLCBLC2 QKSD001F-Failed-
         CLOSE-attempt...'
36.        UPON CONSOLE
37.      MOVE QKSD001F-STATUS TO IO-STATUS
38.      PERFORM DISPLAY-IO-STATUS
39.    END-IF
40.    EXIT.
```

В этом фрагменте просматривается три паттерна с одним поисковым образом для процедур *QKSD*001*F-GET*, *QKSD*001*F-OPEN*, *QKSD*001*F-CLOSE* соответственно. В этих процедурах множество *B* представлено строками 3–5, 9–13, 15–16, 19–28, 31–40, множество *P* – представляют: в строке 10 – оператор *READ*, в строке 23 – оператор *OPEN*, в строке 35 – оператор *CLOSE*, а множество *V* – представляют строки 6–8 и 14. Строки с номерами 1, 2, 17,18, 29, 30 к поисковому образу паттерна не относятся.

При формальном описании поискового образа паттерна он заключается в операторные скобки *pattern NAME* и *end NAME*, где начальная скобка сдержит ключевое слово *pattern* и имя патерна, а конечная – ключевое слово *end* и имя патерна *NAME*. В связи с тем, что все паттерны, найденные разными аналитиками, складируются в репозитории, необходима сквозная, в пределах всей системы, идентификация имён паттернов. При этом вновь заносимый паттерн проверяется на наличие такого или подобного в репозитории. Если такой или подобный паттерн существует, аналитику предлагается воспользоваться уже зарегистрированным в репозитории именем паттерна и самим паттерном. Поэтому формальный поисковый образ для приведенного паттерна можно представить следующим образом:

Пример 2.

```
1.    pattern NAME.
2.    IF  QKSD001F-STATUS = '00'
3.      SUBTRACT PASS-RESULT FROM PASS-
        RESULT
4.    ELSE
5.      $$ MLINE = 3 MSYMB = 33 $$
6.        ADD 12 TO ZERO GIVING PASS-RESULT
7.        DISPLAY '* CBLCBLC2 QKSD001F-
          Failed-@@ MSYMB = 5 @@-attempt...'
8.          UPON CONSOLE
9.        MOVE QKSD001F-STATUS TO IO-STATUS
10.         PERFORM DISPLAY-IO-STATUS
11.     $$ MLINE = 1 MSYMB = 6 $$
12.   END-IF
13.   EXIT.
14.   end NAME.
```

В представленном поисковом образе паттерна при описании множества *P* (*@@ MSYMB* = 5 *@@* в строке 10) и подмножеств *V* (строки 5 и 11) использованы ключевые слова *MLINE* (указывает максимальное количество линий в данном параметре или подмножестве операто-

ров), и *MSYMB* (указывает максимальное количество символов в строках данного параметра или подмножества операторов). В случае отсутствия любого или обоих ключевых слов, по умолчанию принимается значение длины, равное единице. Например, приведенные в примере 3 описания подмножества *V* будут равнозначны:

Пример 3.

$$ $$ и $$ *MLINE* =1 *MSYMB* = 1 $$
$$ *MSYMB* = 24 $$ и $$ *MLINE* =1 *MSYMB* = 24 $$

Минимальное количество линий и строк в *P* и *V* может равняться нулю, т.е. некоторые параметры или подмножества операторов в некоторых экземплярах паттернов могут отсутствовать. Тогда запись @@ *MSYMB* = 0 @@ или @@ *MLINE* = 0 @@ обозначает отсутствие подмножества *P* и может быть вообще опущена. Аналогично и для множества *V*.

Отметим, что поисковый образ паттерна не может начинаться и заканчиваться множествами *P* и *V*, т.е. эти множества всегда должны находиться внутри множества *B*.

Из приведенных примеров видно, что ключевые слова для *P* заключаются в операторные скобки @@, а для подмножеств *V* – в операторные скобки $$. Именно по этим операторным скобкам и правилам описания поисковых образов паттернов парсер определения паттернов будет осуществлять их поиск в модулях анализируемой системы. Поисковые образы определяются аналитиком и в формализованном виде, описанном выше, заносятся в специальный файл парсера определения паттернов, откуда парсер их выбирает по имени и осуществляет поиск паттернов по этим поисковым образам во входной программе или в анализируемой системе в целом. Отлаженные паттерны складируются в репозиторий, где системные аналитики выполняют их обобщение и классификацию.

Поиск паттернов осуществляется по следующему алгоритму:

- из поискового образа выбирается часть множества *B* до первой операторной скобки (@@ или $$);
- осуществляется наложение и продвижение выделенной части множества *B* по входной цепочке с целью их поиска в исходной программе;
- в случае обнаружения выделенной части множества *B* во входной цепочке выбираются текущие значения *MLINE* и *MSYMB* из заданного поискового образа и выделяется следующая часть множества *B* до следующей операторной скобки (@@ или $$);
- осуществляется наложение и продвижение вновь выделенной (следующей) части множества *B* по входной цепочке с целью ее поиска в исходной программе. При этом контролируют, чтобы количество продвинутых линий и символов исходной программы не превышало значений *MLINE* и *MSYMB*, определенных в предыдущем шаге. Если количество продвинутых линий и символов исходной программы превысило значения *MLINE* и *MSYMB*, то поиск паттерна трактуется как неудачный. В противном случае поиск продолжается с третьего пункта;
- поиск паттерна считается успешным в случае обнаружения во входной программе заданного множества *B*, когда все найденные в программе варианты *P* и *V* во множестве *B* по размерам не должны превышать заданных в поисковом образе значений *MLINE* и *MSYMB*.

Выделение отдельных программных блоков

Эта операция выполняется путем отслеживания в программном модуле всех операторов, выполняющих функции операторных скобок для внутримодульных императивных программных блоков. Кроме того, аналитику путем визуального просмотра предоставляется возможность определять блоки, по его мнению, имеющие логическую завершенность.

При идентификации внутримодульных блоков выполняется отслеживание попадания в тело блока операторов вызова других внутримодульных блоков этого модуля. Такое отслеживание позволяет построить дерево внутримодульных вызовов, что существенно облегчит работу аналитика при выполнении анализа всех логических ветвей программы. При этом

выявляются участки «мертвого» кода на внутримодульном уровне, которые удаляются из системы и не подвергаются дальнейшему анализу и реинжинирингу.

Выделение отдельных, логически завершенных, программных блоков выполняется в двух режимах: автоматическом и ручном.

В ***автоматическом режиме*** парсер выделения программных блоков отслеживает в модуле операторы, выполняющие функции операторных скобок программных блоков в императивной части программы. Например, в языке *COBOL* указанные операторные скобки отображаются именами процедур, стоящих в операторах *PERFORM*, и операторами *EXIT*, *GO TO*; в языке *FORTRAN* такими операторами являются *SUBROUTINE* (*FUNCTION*) и *RETURN* (*STOP*); в языке *PL*/1 эта роль отведена операторам *PROCEDURE* и *RETURN*.

В процессе сканирования по программе парсер выделения программных блоков выполняет нумерацию строк исходного кода. При обнаружении пары операторных скобок блока в таблицу программных блоков модуля заносится имя программного блока, взятого из начальной операторной скобки, а также номера строк начальной и конечной операторных скобок, определяющих границы тела программного блока. Отметим, что в некоторых языках программирования, например *Assembler*, программные блоки могут состоять из нескольких частей, расположенных в разных местах программного модуля. В этом случае каждая часть программного блока в таблице программных блоков будет занесена с указанием границ каждой части отдельно.

В примере 4 приведен фрагмент программы на языке *COBOL*, в котором объявлены процедуры *F*19, *F*98*DT*, *F*98*ER*, *F*20. Слева показана нумерация строк, выполняемая парсером выделения программных блоков. В табл. 1 показан пример заполнения таблицы программных блоков для приведенного фрагмента программы.

Пример 4.

```
48  F19.
. . .
57  PERFORM F98DT
58  MOVE DT1F-EMCROH TO DT3T-EMCROH
59  PERFORM F98ER
60  EXIT.
. . .
102  F98DT.
103  MOVE 'Y' TO WSER-OPEN
104  MOVE SPACES TO DV00-TUDIV0
105  EXIT.
. . .
200  F98ER.
201  GO TO F20.
. . .
271  F20.
272  EXIT.
```

Таблица 1

Имя программного блока	Номер строки начальной операторной скобки блока	Номер строки конечной операторной скобки блока
*F*19	48	60
*F*20	271	272
*F*98*DT*	102	105
*F*98*ER*	200	201

После определения границ внутримодульных блоков выполняется отслеживание операторов вызова программных блоков в модуле (операторы *PERFORM*) и операторов передачи управления (*GO TO*). Внешние вызовы в этом случае не учитываются, так как они обрабатываются при построении дерева вызовов модулей системы [2].

При выявлении оператора вызова программного блока или оператора передачи управления по номеру строки данного оператора выполняется поиск в таблице программных блоков с целью установления, к какому блоку относится выявленный оператор. Установленное в таблице программных блоков имя вызывающего блока заносится в таблицу внутренних вызовов, если оно еще не занесено в таблицу. В эту же строку таблицы внутренних вызовов помещается имя вызываемого блока. Если имя вызывающего блока уже занесено в таблицу внутренних вызовов, то в его строку заносится только имя вызываемого блока. В случае вызова с указанием диапазона выполнения программных блоков (формат *THRU*) в строку заносятся имена всех вызываемых блоков, попадающих в заданный диапазон выполнения про-

граммных блоков. Эти имена определяются путем анализа границ внутримодульных блоков в таблице программных блоков, т.е. из таблицы программных блоков выбираются имена внутримодульных блоков, границы которых вкладываются в диапазон выполнения программных блоков.

Имя программного блока, которому передается управление с помощью оператора *GO TO*, определяется путем идентификации номера строки, которой передается управление, и установления посредством анализа границ программных блоков в таблице этих блоков, к какому именно блоку она относится.

Заполнение таблицы внутренних вызовов для примера 4 показано в табл. 2.

Таблица 2

Имя вызывающего программного блока	Имя вызываемого программного блока 1	Имя вызываемого программного блока 2	. . .	Имя вызываемого программного блока *N*
*F*19	*F*98*DT*	*F*98*ER*		
*F*98*ER*	*F*20			

Таблица внутренних вызовов отображает дерево вызовов программных блоков внутри модуля.

Параллельно с заполнением таблицы внутренних вызовов строится временная таблица активных областей программы. При всякой идентификации вызывающего или вызываемого программного блока в таблице активных областей отмечаются границы этого программного блока. Отметка границ в таблице активных областей выполняется следующим образом: если в таблице активных областей уже существует отметка, включающая в себя пространство программного блока, то никаких изменений в таблице активных областей не происходит. Если пространство программного блока примыкает к какой-либо границе отметки, уже существующей в таблице активных областей или перекрывает ее, то эта граница изменяется с целью включения пространства программного блока в отметку таблицы активных областей. Если же пространство программного блока одной границей примыкает к какой-либо границе одной отметки в таблице активных областей или перекрывает ее, а другая граница примыкает к какой-либо границе другой отметки в таблице активных областей или перекрывает ее, то эти отметки объединяются. Если же пространство программного блока не примыкает к границам отметок в таблице активных областей, то в этой таблице строится новая отметка с соответствующими границами программного блока.

По окончании выделения логически завершённых программных блоков, выполняется анализ информации, отмеченной в таблице активных областей. Если обнаружены участки анализируемого программного модуля, не попавшие в таблицу активных областей, то аналитику выдается сообщение о наличии в модуле участков «мертвого» кода на внутримодульном уровне с указанием их границ, а если участок «мертвого» кода является программным блоком, то сообщается его имя.

Как и в случае с «мертвым» кодом на внешнемодульном уровне [2], обнаруженные участки «мертвого» кода на внутримодульном уровне удаляются из системы и не подвергаются дальнейшему анализу и реинжинирингу.

Ручной режим выделения отдельных, логически завершенных, программных блоков предполагает предварительный просмотр анализируемого модуля аналитиком, с целью определения в нем неформализованных, логически завершенных программных блоков. Выделенные блоки отмечаются специальными начальными и конечными метками блоков. Далее процесс построения дерева вызовов программных блоков внутри модуля происходит по описанному алгоритму автоматического режима. В этом случае в таблицу программных блоков и таблицу внутренних вызовов будут заноситься как программно определенные блоки, так и блоки, отмеченные аналитиком. Последние будут именоваться заранее определяемыми стандартными именами, например: *BLOK*1, *BLOK*2, *BLOK*3, ..., *BLOKn*.

Построенное в ручном режиме дерево вызовов программных блоков будет более детально

отображать логику передачи управления внутри анализируемого модуля. Кроме того, аналитик имеет возможность менять границы отмечаемых блоков, и повторно строить дерево вызовов программных блоков. Сравнивая полученные деревья внутримодульных вызовов, можно выбрать такое, которое наиболее наглядно отражает бизнес-процесс, положенный в основу функционирования анализируемого модуля.

Построение дерева внутримодульных вызовов существенно облегчает работу аналитика, а также позволяет осуществлять контроль проводимого анализа всех логических ветвей программы.

Заключение. Изложенная методика и алгоритмы автоматизации определения паттернов в модулях *legacy*-систем, способствующие решению проблемы обобщения и унификации создаваемых БП, а также их классификации в рамках конкретной *legacy*-системы или ряда таких систем, относящихся к одному типу, позволили получить следующие результаты:

- разработан механизм обобщения и унификации создаваемых БП, а также их классификации в рамках анализируемых *legacy*-систем;
- предложены методики построения специальных структурированных хранилищ БП (репозитории); инструментального интерфейса для работы с репозиториями как в автоматизированном, так и в интерактивном режимах;
- разработан алгоритм построения дерева внутримодульных вызовов для контроля и анализа всех логических ветвей программы;
- определены требования к парсерам и поиска их в модулях *legacy*-систем, построения таблиц внутренних вызовов и таблиц условий выполнения заданий при автоматизации построения БП на этапе проведения реинжиниринга программных *legacy*-систем.

1. *Реализация* реинжиниринга программных *legacy*-систем / А.В. Анисимов, В.В. Белодед, Н.Д. Пашковец и др. // УСиМ. – № 6. – 2008. – С. 40–48.
2. *Автоматизация* построения бизнес-правил в процессе реинжиниринга программных *legacy*-систем на этапе анализа их функциональных структур / В.И. Гриценко, А.В. Анисимов, Н.Д. Пашковец и др. // УСиМ. – 2009. – № 3. – С. 56–64.
3. *Построение* бизнес-правил для *SQL*-вложений и *JCL*-заданий / В.И. Гриценко, А.В. Анисимов, Н.Д. Пашковец и др. // УСиМ. – 2009. – № 4. – С. 26–33.

Поступила 14.08.2009

Тел. для справок: (044) 526-4187 (Киев)

Окончание статьи И.П. Атаманюка

2. *Васильев Б.В.* Прогнозирование надежности и эффективности радиоэлектронных устройств. – М.: Сов. радио, 1970. – 335 с.
3. *Пугачев В.С.* Теория случайных функций и ее применение. – М.: Физматгиз, 1962. – 720 с.
4. *Кудрицкий В.Д.* Прогнозирующий контроль радиоэлектронных устройств. – Киев: Техніка, 1982. – 168 с.
5. *Атаманюк И.П.* Алгоритм реализации нелинейной случайной последовательности на базе ее канонического разложения // Электронное моделирование. – 2001. – № 5. – С. 38–46.
6. *Атаманюк И.П.* Полиномиальный алгоритм оптимальной экстраполяции параметров стохастических систем. // УСиМ. – 2002. – № 1. – С. 16–19.
7. *Атаманюк И.П.* Алгоритм экстраполяции нелинейного случайного процесса на базе его канонического разложения // Кибернетика и системный анализ. – 2005. – № 2. – С. 131–138.
8. *Parzen E.* On the estimation of probability density function and the mode // Ann. Math. Stat. – 1962. – **33**. – P. 1065–1076.
9. *Епанечников В.А.* Непараметрическая оценка многомерной плотности вероятности // Теория вероятностей и ее применение. – 1969. – № 1. – С. 156–161.
10. *Рубан А.И.* Непараметрические процедуры сглаживания результатов эксперимента // Сб.: Системы управления, вып. 2. – Томск: Изд–во Томского ун-та, 1977. – С. 46–54.
11. *Кудрицкий В.Д.* Фильтрация, экстраполяция и распознавание реализаций случайных функций. – К.: ФАДА, ЛТД, 2001. – 176 с.

Поступила 25.10.2009

Тел. для справок: (0512) 218-303, (098) 797-1234 (Николаев)
E-mail: *atamanyuk_igor@mail.ru*