УДК 681.31

Алгоритмы стандартизации исходного кода.

С.А. Кондаков, Н.А. Кудинов

В статье рассматривается задача корпоративной стандартизации исходного кода программного обеспечения, и вводятся показатели удовлетворения исходного кода требованиям корпоративной стандартизации, рассматривается алгоритм построения системы корпоративных стандартов.

В современном мире разработка программного (ПО) обеспечения превратилась в одну из самых дорогостоящих индустрий, и любые узкие места в технологическом процессе его создания могут привести к нежелательным результатам. Удлинение сроков разработки ПО чревато удорожанием конечного продукта. Ошибки, не выявленные в ходе тестирования ПО, приводят к снижению надежности и затягиванию сроков его внедрения. Поэтому актуальность задачи корпоративной стандартизации, позволяющей сократить время разработки программного обеспечения и повысить его надежность, весьма высока.

Корпоративные стандарты написания исходного кода. Введение корпоративных стандартов кодирования [1,2,3,4] позволяет сократить время на разработку и поддержку программного продукта. Это достигается за счет следующих факторов:

1) не приходится отвлекать главных разработчиков для обучения новых сотрудников;

2) изучение стандартизированных частей продукта проходит быстрее;

3) за счет стандартизированного кода не допускаются типичные ошибки разработчиков;

4) тратится меньше времени на решение проблем, возникающих в процессе написания кода (как назвать класс, как назвать переменную, куда ее поместить и т.п.).

Автоматизированные средства контроля. Сложно или даже невозможно визуально проконтролировать весь текст программного обеспечения. Поэтому актуальна задача автоматизации этой процедуры.

Помимо контроля соблюдения стандартов есть и другие цели анализа исходного кода ПО. Рассмотрим эти цели и средства автоматизации, используемые для этих целей для того, чтобы понять и проследить тенденции в области анализа исходного кода.

Цели анализа исходного кода. Цели анализа исходного кода можно разделить на следующие три группы:

1. вычисление метрических показателей сложности программного обеспечения;
2. выявления логических ошибок в программе;

3. контроль соблюдения стандартов.

В настоящее время доступны программы для автоматического анализа исходного кода, которые могут оказать помощь разработчикам и верификаторам ПО. Такие программы представлены в таблице 1.

Средства статистического анализа исходного кода. Таблица 1

Наименование	Вычислимые метрики	Логические Ошибки	Проверяемые Стандарты	Поддерживаемые языки
Checkstyle	-	-	+	java
Jtest	+	+	+	java
C++Test	-	+	-	C++
CodeWizard	-	+	+	C++
Insure++	+	+	-	C++
Pc-lint	-	+	-	C++
Kratau metrix	+	-	-	C++, Java

Анализ текущего положения (таблица 1) показывает, что, практически все средства анализа исходного кода разработаны для таких популярных языков как Java и C++. И эти средства не предоставляют возможности настройки их для работы с другими языками. Таким образом, актуальная задача построения системы контроля корпоративных стандартов для языка PL/SQL не решена.

Рассмотрим, что представляют собой корпоративные стандарты написания исходного кода. В качестве примера рассмотрим венгерскую нотацию [6] для языка C, поскольку венгерская нотация создавалась специально для него, а также предложенные авторами стандарты для языка PL/SQL.

Венгерская нотация представляет собой строгий набор правил для составления имен переменных и процедур. Нотация получила название "венгерской" потому, что ее изобрел Чарльз Симони, венгр по происхождению.

Имена составляются из трех частей : один или несколько префиксов, базовый тип и спецификатор т.е <имя переменной>:=:(**префикс**)*(**базовый тип)(спецификатор)**.

Префиксы описывают, как используется данная переменная. Префиксы стандартизованы (см. следующую таблицу и MSDN для более подробного списка):

Префиксы в венгерской нотации. Таблица 2.

Префикс	Значение
A	Массив
C	Счетчик (например, в количестве записей, букв и т.д.)

D	Разность между двумя элементами одного типа
E	Элемент массива
G	Глобальная переменная
H	Handle
I	Индекс массива
M	Переменная уровня модуля
P(lp,np)	Указатель (длинный или короткий указатель для Intel'овских машин)

Базовый тип указывает на тип переменной. Обычно это не стандартный, а некоторый более абстрактный тип, такой как окна, области экрана, шрифты и т.д. Для каждой переменной может существовать только один базовый тип. В следующей таблице приведен пример базовых типов, которые можно было бы использовать при написании текстового процессора:

Пример базовых типов для венгерской нотации. Таблица 3.

Базовый тип	Значение
wn	Окно
Src	Область экрана
Fon	Шрифт
Ch	Литера (не в смысле C, а в смысле структуры данных, которая используется в данном текстовом процессоре для представления буквы)
pa	Абзац (параграф)

Спецификаторы. Последней частью венгерской нотации является спецификатор (qualifier). Спецификатор - это содержательная часть имени, т.е. то, что было бы именем, если бы не использовалась венгерская нотация.

В дополнение к тем спецификаторам, которые будут созданы лично Вами, венгерская нотация имеет несколько стандартных спецификаторов, которые предназначены для типового оформления некоторых стандартных ситуаций:

Стандартные спецификаторы в венгерской нотации. Таблица 4.

Спецификатор	Значение
Min	Самый первый элемент любого массива или списка другого рода (т.е. абсолютная точка отсчета)
First	Первый обрабатываемый элемент массива (относительно данной операции)
Last	Последний обрабатываемый элемент массива (антоним для First)
Lim	Верхняя граница для обрабатываемых элементов массива. Lim является антонимом First и не является действительным индексом для массива. Чаще всего Lim = Last + 1
Max	Самый последний элемент массива. Чаще всего Max используется для ссылки на массив в целом, а не для операций с элементами

Например, индекс массива окон определяется как iawn: здесь i –префикс (индекс массива), a-префикс (массив), wn-базовый тип (окно).

Достоинства венгерской нотации. У венгерской нотации те же достоинства, что и у любой другой четко оговоренной стандартной договоренности об именовании переменных. Например, стандартизация такого большого количества имен позволяет тратить меньше времени на запоминание переменных в своей программе или разбор чужой программы. Кроме того, венгерская нотация носит достаточно общий характер, что делает возможным ее использование в самых различных проектах.

Венгерская нотация позволяет проверять некоторые операции уже в процессе чтения программы (например, вырезка cpaReformat[i] скорее всего синтаксически неправильна, так как первая буква имени не показывает, что переменная является массивом). Таким образом, венгерская нотация может помочь при работе в слабо типизованном окружении. В частности, при программировании под Windows разработчикам приходится то и дело явно преобразовывать типы данных, поэтому используемые соглашения могут помочь справиться с этой ситуацией.

Недостатки венгерской нотации. Некоторые широко распространенные версии венгерской нотации используют следующее упрощение: в качестве базовых типов используются только стандартные типы. Такой вариант мало интересен, поскольку с

базовыми типами программисты чаще всего справляются без использования каких-либо соглашений. Значительно содержательнее использование венгерской нотации при работе с абстрактными типами данных.

Другая проблема венгерской нотации заключается в том, что она смешивает значение переменной и ее представление. Так, при объявлении целой переменной нам не хотелось бы менять имя переменной только потому, что мы изменили ее тип на длинное целое. А именно это нам и приходится делать при использовании венгерской нотации.

Наконец, последний упрек в адрес венгерской нотации заключается в том, что программист по-прежнему может создать абсолютно ничего не говорящее имя. В результате, все, что мы получаем взамен семантики переменной - это точное определение ее типа, что не так уж и много. К счастью, данной проблемы всегда можно избежать, если для каждой переменной использовать спецификаторы.

После рассмотрения венгерской нотации перейдем к рассмотрению правил корпоративной стандартизации для языка PL/SQL.

Правила корпоративной стандартизации языка PL/SQL, предложенные авторами, приведены в таблице 5.

Правила корпоративной стандартизации для PL/SQL . Таблица 5.

Правило корпоративной стандатизации исходного кода
$\forall x \in$ Q1, где Q1 - множество зарезервированных слов SQL и PL/SQL. F1(x)={0, если ACSII(‘a’)<=ASCII(xi)<=ASCII(‘z’) для $\forall i$: 1<=i<=length(x), 1 иначе Все служебные зарезервированные слова SQL и PL/SQL пишутся строчными (маленькими) буквами.
$\forall x \in$ Q2, где Q2 - множество идентификаторов таблиц и полей в таблицах и курсорах. F2(x)={0, если ACSII(‘A’)<=ASCII(xi)<=ASCII(‘Z’) для $\forall i$: 1<=i<=length(x), 1 иначе Идентификаторы таблиц и полей в таблицах и курсорах пишутся прописными (заглавными) буквами.
$\forall x \in$ Q3, где Q3 - множество идентификаторов типов данных, определенных пользователем. F3(x)={0 если substr(x,1,5) = ‘type_’, 1 иначе

Идентификаторы типов данных, определенных пользователем, начинаются с префикса (type_).
$\forall x \in$ Q4, где Q4 - множество идентификаторов локальных констант. F4(x)={0, если substr(x,1,2) = ‘c_’, 1 иначе Идентификаторы локальных констант начинаются с префикса (c_)
$\forall x \in$ Q5, где Q5 - множество идентификаторов глобальных констант. F5(x)={0, если substr(x,1,3) = ‘gc_’, 1 иначе Идентификаторы глобальных констант начинаются с префикса (gc_).
$\forall x \in$ Q6, где Q6 - множество идентификаторов локальных переменных. F6(x)={0, если substr(x,1,2) = ‘v_’, 1 иначе Идентификаторы локальных переменных начинаются с префикса (v_)
$\forall x \in$ Q7, где Q7 - множество идентификаторов глобальных переменных. F7(x)={0, если substr(x,1,3) = ‘gv_’, 1 иначе Идентификаторы глобальных переменных начинаются с префикса (gc_).
$\forall x \in$ Q8, где Q8 - множество идентификаторов локальных курсоров. F8(x)={0 если substr(x,1,4) = ‘curs_’, 1 иначе Идентификаторы локальных курсоров начинаются с префикса (curs_)
$\forall x \in$ Q9, где Q9 - множество идентификаторов глобальных курсоров. F9(x)={0, если substr(x,1,5) = ‘gcurs_’, 1 иначе Идентификаторы глобальных курсоров начинаются с префикса (gcurs_).
$\forall x \in$ Q10, где Q10 - множество идентификаторов параметров процедур и функций. F10(x)={0, если substr(x,1,2) = ‘p_’, 1 иначе Идентификаторы параметров процедур и функций начинаются с префикса (p_).

$\forall x \in$ Q11, где Q11 – множество исходных кодов процедур и функций . F11(x)={0 если в теле есть вызов KRN_Trace.Start_Proc, KRN_Trace.End_Proc, Exception when others then krn_trace.Error_Handler(), 1 иначе Тело каждой процедуры и функции должно начинаться вызовом пакетной процедуры KRN_Trace.Start_Proc с именем соответствующей процедуры в качестве аргумента и заканчиваться вызовом пакетной процедуры KRN_Trace.End_Proc. Каждая процедура, функция или триггер должны содержать в своем теле секцию обработки исключений (exception), в которой осуществляется вызов пакетной процедуры KRN_Trace.Error_Handler
$\forall x \in$ Q12, где Q12 – множество исходных кодов триггеров. F12(x)={0 если в теле есть вызов KRN_Trace.Start_Trigger, KRN_Trace.Start_Trigger, 1 иначе Тело каждого триггера должно начинаться вызовом пакетной процедуры KRN_Trace.Start_Trigger с именем соответствующего триггера в качестве аргумента и заканчиваться вызовом пакетной процедуры KRN_Trace.End_Trigger.
$\forall x \in$ Q13, где Q13 – множество исходных кодов процедур, функций или триггеров, в которых используется оператор return . F13(x)={0, если в теле есть вызов KRN_Trace.End_Proc или KRN_Trace.End_Trigge перед оператором return, 1 иначе Перед обращением к оператору return в теле процедуры, функции или триггера должен быть вызов пакетной процедуры KRN_Trace.End_Proc или KRN_Trace.End_Trigger.
$\forall x \in$ Q14, где Q14 – множество исходных кодов курсоров. F14(x)={0, если в теле не используются переменные , 1 иначе При объявлении курсоров не допускается использовать определенные вне их тела переменные - передача значений осуществляется только через их параметры (аргументы)
$\forall x \in$ Q15, где Q15 – множество исходных кодов процедур и функций.

F15(x)={0, если в теле не используются переменные , 1 иначе Внутри процедур и функций не допускается использовать определенные вне их тела переменные, кроме глобальных переменных пакета (gv_.., gc_..) - передача значений осуществляется только через их параметры (аргументы)
$\forall x \in$ Q16, где Q16 – множество исходных кодов процедур и функций и триггеров. F16(x)={0, если в теле нет оператора select .. into , 1 иначе В триггерах, пакетах, процедурах и функциях все операторы SELECT желательно выполнять в виде курсоров

При анализе отличий предложенной авторами системы корпоративных стандартов от представленной выше венгерской нотации следует выделить тот факт, что венгерская нотация состоит только из правил наименования. Предложенные правила корпоративной стандартизации языка PL/SQL включают требования, улучшающие читабельность программ, а также правила обработки ошибок и другие правила, не ограничивающиеся одним лишь требованием правильного именования.

Для решения задачи корпоративной стандартизации исходного кода введем формальную модель языка с корпоративными ограничениями и предложим показатели удовлетворения модуля корпоративным стандартам.

Определение. Системой корпоративной стандартизации SCS языка L будем называть восьмерку SCS =<L, Q , E, A, T, P, M, H>, где

L -язык, используемый при написании информационной системы;

Q - множество типов объектов проверки: Q={Q_i| i=1,…,N – порядковый номер типа объекта проверки};

E - множество типов ошибок корпоративной стандартизации: E=$\bigcup_{Qi} E_{jQi}$, j=1,…,k_{Qi}. У каждого типа объекта проверки Q_i может контролироваться более одного типа ошибок;

A - множество объектов проверки (фрагментов модулей исходного кода): A=$\bigcup_{Qi} A_{Qi}$;

T - отображение цепочек в подмножество множества объектов проверки. T : L$\rightarrow$ρ(A), где ρ(A) – множество всех подмножеств множества A ;

P - множество правил корпоративных стандартов кодирования $\forall$ p: $p_{E_{Qi}}(a) = \{0,1\}$ a$\in$A

M - оценка соответствия требованиям корпоративной стандартизации произвольной цепочки из L вводится как: L $\rightarrow$ R т.е M(l)=m где l $\in$ L, m $\in$ R.

H - функция корректировки H: L$\rightarrow$L такая, H(l)=l’ что M (l) $\geq$ M (l’) где l, l’ $\in$ L

Таким образом, задача, поставленная в данной работе – формализация процесса контроля за выполнением требований корпоративной стандартизации в процессе разработки программного обеспечения, включает:

- Выделение множества типов ошибок корпоративной стандартизации.
- Разработку формальных правил кодирования или требований, предъявляемых к исходному программному коду, что выражается в составлении системы правил стандартизации – $\{P_j\}$.
- Алгоритмизацию процедур контроля за соблюдением требований корпоративных стандартов, которые выражаются в вычислении функции оценки качества их удовлетворения.
- Алгоритмизация исправления типовых нарушений требований стандартизации, выражаемая в определении функции корректировки ошибок H: $H(\alpha) = \alpha'$ такой, что $M(\alpha') <= M(\alpha)$, где α и $\alpha' \in L$, а программа при этом не теряет своей функциональности.

Для пояснения формальной системы приведем пример.

Пример формальной системы:

L–{“ declare v_FileName char(30); begin print(FileName) end;”, ” declare cursor files is select * from FileList; FileName char(30); begin print(FileName) ; end;”}

Q -{переменная, курсор}

E -{ошибка наименования переменной, ошибка наименования курсора}

A–{v_FileName, FileName, files}

T-{<“ declare cursor file is select * from FileList; FileName char(30); begin print(FileName) ; end;”,{FileName, files}>,<”declare v_FileName char(30) begin print(FileName) end;, {v_FileName}>}”

P-{ наименование переменной должно начинаться с префикса v_, наименование курсора должно начинаться с префикса curs_}

M(“declare cursor file is select * from FileList; FileName char(30); begin print(FileName) ; end;”) = 2

M(“declare v_FileName char(30); begin print(v_FileName) end;”) = 0

Рассмотрим схему, поясняющую предложенную нами модель корпоративной стандартизации рис 1.

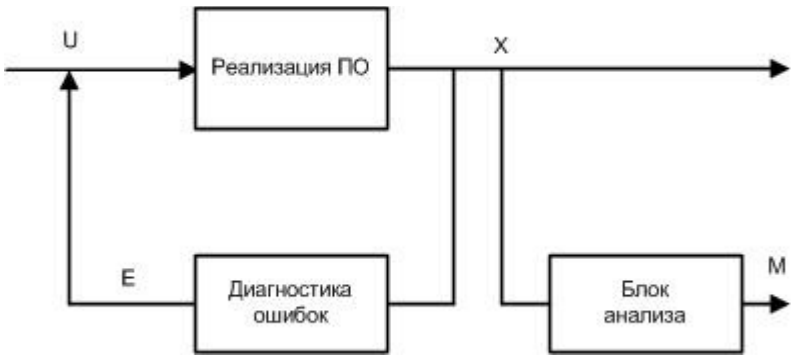

Рисунок 1. Схема реализации ПО

где U = {u1,u2,u3,…} – требования (ТЗ), X={x1,x2,x3,…} – исходный код, E ={e1,e2,e3,..} ошибки стандартизации (ТЗ исправления ошибок). Задача управления процессом корпоративной стандартизации заключается в том, чтобы в любой момент времени вектор ошибок был ограниченным, т.е |E| < e.

Указанное условие достижимо не для всякого набора P правил корпоративной стандартизации. Правила могут быть невыполнимы, и тогда система не будет устойчивой.

Определение. Группа правил pi называется невыполнимой, если не $\exists$ x : P1&p2&pn = True

Приведем алгоритм проверки выполнимости группы правил.

Пусть {pi} – правила невыполнимость которых мы проверяем. Рассмотрим следующий алгоритм, который позволяет проверить набор правил на выполнимость.

1. НАЧАЛО j:=N.
2. ПРОВЕРКА: Если $\exists x \in X_j : p_j(x) \& .. \& p_N(x) = True$ то ИТЕРАЦИЯ иначе НАБОР НЕВЫПОНИМ.
3. ИТЕРАЦИЯ: j:=j-1. Если j=0, то НАБОР ВЫПОЛНИМ иначе ПРОВЕРКА..
4. НАБОР НЕВЫПОЛНИМ: КОНЕЦ.
5. НАБОР ВЫПОЛНИМ: КОНЕЦ.
6. КОНЕЦ.

где $X_j = S_{j-1} \cap S_j$, $S_j = \{x : p_j(x) = True\}$

Таким образом, данный алгоритм позволяет выявить противоречивые группы правил или убедиться в непротиворечивости группы правил.

Перейдем к рассмотрению алгоритмов построения системы корпоративной стандартизации. Для этого введем определения реляционной и xml-модели данных и рассмотрим вопросы отображения реляционной базы данных в xml-документы. Такое отображение используется при построении системы проверки корпоративных стандартов т.к проектные данные хранятся в реляционной БД, а авторами по ряду причин принято решение использовать XML-документы для обмена информацией между модулями системы контроля корпоративных стандартов.

Реляционная модель

Определение. Реляционной схемой отношения будем называть конечное множество атрибутов A_1, A_2, …, A_n и обозначать R.

Определение. Схемой реляционной базы данных будем называть множество схем реляционных отношений, и обозначать $R(R_1, R_2, ,..)$.

Множество реляционных схем R будем обозначать буквой ℜ

Пусть $D_1,D_2,...,D_N$ - некоторые множества, которые мы будем в дальнейшем называть доменами.

Определение. Реляционным отношением r будем называть множество упорядоченных n-ок. $< d_1, d_2 ,..., d_n >$ где $d_1 \in D_1, d_2 \in D_2,.., d_n \in Dn$.

XML модель

Определение. Схемой XML-Документа будем называть пару $X= <\{ A_i \mid i=1..m\}, < A_i, A_j> >$, т.е. множество атрибутов (наименований узлов) и отношение на этом множестве.

Множество схем XML-документов будем обозначать ℵ

Определение. XML-документом будем называть множество пар $x=\{< d_i, d_j > \mid d_i \in Di, d_j \in Dj\}$

Рассмотрим вопрос отображения реляционных схем в xml-схемы .

Утверждение 1.

Любое реляционное отношение r со схемой R может быть отображено в xml-документ x со схемой X, т.е. существует отображение R2X: ℜ->ℵ(Отображение множества реляционных схем в множество схем XML-документов будем обозначать R2X).

Предложим алгоритм такого отображения (R2X). Документ строится согласно следующему алгоритму:

НАЧАЛО

Шаг1: Вводим атрибуты A_1, A_2, …, A_n – из реляционной модели $R = \{ A_1, A_2, …, A_n \}$ и атрибут AR – наименование отношения r тогда $X=\{\{ A_1, A_2, …, A_n,AR\}, <AR,Ai> \mid i =1..n\}$
Шаг2: Каждый кортеж отношения $r= (d_1, d_2 ,..., d_n)$ отображается в $\{ <AR, d_1>,<AR, d_2 >,..,<AR, d_n >\}$.
Множество всех кортежей отобразятся в xml-документ.

КОНЕЦ

Рассмотрим вопрос отображения xml-схемы в реляционные схемы.

Утверждение 2.

Любой xml-Документ x со схемой X может быть отображен на реляционное отношение r со схемой R, т.е. существует отображение X2R: X->R (Отображение множества реляционных схем в множество схем XML-документов будем обозначать R2X).

Предложим алгоритм такого отображения (X2R). Документ строится согласно следующему алгоритму:

НАЧАЛО

Шаг1: Схему R отношения строим следующим образом R={ID, Name, Value, ParentID}
Шаг2: Каждому отношению <di,dj> ставим в соответствие 2 кортежа <IDi, Name di, Value di, null>, <IDj, Name dj, Value dj, IDi>

КОНЕЦ

Таким образом мы доказали, что любое реляционное отношение может быть отображено в XML-документ и любой XML-документ может быть отображен в реляционное отношение, что позволяет нам использовать XML-документы для хранения информации из реляционных баз данных и сохранять XML-данные в реляционной базе.

В данной статье не рассматриваются вопросы оптимизации представления XML-документов в базе данных и таблиц в XML-документах. С этими вопросами можно ознакомиться, например в [12].

Рассмотрим вопросы, касающиеся выявления ошибок корпоративной стандартизации цепочки языка. Рассмотрим следующие понятия.

Дерево вывода цепочки α языка L

Определение дерева вывода возмем из [7] и будем обозначать его D.

Дерево объектов проверки проекта

Дерево объектов проверки проекта PT = $\{\{A_i\}, <Ai,Aj>\}$ где $Ai \in A$.

Дерево проверки корпоративных стандартов

Пусть D – дерево вывода (дерево разбора) цепочки α в КС-грамматике G(A)=(N,Z,P,A). Деревом корпоративной стандартизации будем называть дерево ST, получаемое из дерева D приписыванием к вершинам d вершин с пометкой из множества E, если вершина d не удовлетворяет правилам корпоративной стандартизации.

Теорема. Пусть G=(N,Z,P,S), - КС-грамматика. E - множество ошибок корпоративной стандартизации - цепочки α тогда и только тогда, - когда в G существует дерево корпоративной стандартизации ST цепочки α с листьями E.

Необходимость. Пусть E - множество ошибок корпоративной стандартизации цепочки α. Покажем, что существует дерево корпоративной стандартизации ST. Т.к α - цепочка в G, то существует дерево вывода этой цепочки, а, следовательно, приписав к этому дереву листья из E для соответствующих вершин, получим дерево корпоративной стандартизации, т. е дерево корпоративной стандартизации существует.

Достаточность. Пусть ST – дерево ошибок корпоративной стандартизации цепочки α. Покажем, что E – множество ошибок корпоративной стандартизации цепочки α. Это непосредственно следует из определения дерева корпоративной стандартизации .

Таким образом, мы доказали, что для того, чтобы найти ошибки корпоративной стандартизации цепочки необходимо построить дерево корпоративной стандартизации и его вершины из множества E будут являться множеством ошибок данной цепочки.

Введем показатели стандартизации.

Метрические показатели стандартизации исходного кода

Метрические показатели стандартизации исходного кода. Авторами предложены следующие показатели:

- СУМ: В ее основе лежит идея оценки качества суммой ошибок несоответствия корпоративным стандартам - $M(X) = \sum_i^n \sum_j^{\#T(X)} Pi(xj)$;
- СУМО: Относительная оценка. СУМ разделенная на количество операторов-$M(X) = \sum_i^n \sum_j^{\#T(X)} Pi(xj) \, 1/\#T(X)$;
- ВСУМ: Взвешенная оценка. Ошибкам присваивается некоторый вес - $M(X) = \sum_i^n \sum_j^{\#T(X)} qiPi(xj)$;
- ВСУМО: ВСУМ разделенная на количество операторов-$M(X) = \sum_i^n \sum_j^{\#T(X)} qiPi(xj) \, 1/\#T(X)$;
- ВКО: Векторная оценка. Вектор ошибок стандартизации $M(X) = (\sum_j^{\#T(X)} P1(xj) , \sum_j^{\#T(X)} P2(xj) \ldots)$.

Введение таких показателей позволяет производить ранжирование программ по некоторому критерию качества.

Произведем классификацию правил корпоративной стандартизации. Такая классификация может быть использована для установления весовых коэффициентов ошибкам нарушения стандартизации.

Классификация правил

Классификация правил. Правила корпоративной стандартизации можно классифицировать следующим образом:

1. Правила, облегчающие чтение программ (упрощают сопровождение);
2. Правила, предотвращающие ошибки (обход критического синтаксиса);
3. Правила обработки ошибок.

Рассмотрим, учитывая приведенные выше утверждения, алгоритм построения системы, предназначенной для контроля за соблюдением корпоративных стандартов.

Технология построения системы

Перейдем к рассмотрению технологии построения автоматизированной системы корпоративной стандартизации.

1. Создание формальной модели SCS языка L. Построение такой модели позволяет выявить объекты предметной области и отношения между ними. Далее следует проверить правила корпоративной стандартизации на выполнимость.
2. Проектирование формата документов описания проекта. Такое структурированное описание проекта позволяет указывать связь между цепочками языка L, что может использоваться для отражения модульной структуры программного обеспечения, указания авторства модуля и указания принадлежности к определенному проекту. Авторами предлагается использовать для описания проекта XML-документы. Использование XML-документов позволяет воспользоваться готовыми и отлаженными средствами для работы с этим форматом.
3. Формирование файла правил корпоративной стандартизации. Авторами предлагается для этих целей также воспользоваться XML-документами.
4. Реализация алгоритма лексического и синтаксического разбора языка L. Данный пункт подразумевает написание парсера языка L. Уточним, что не следует создавать парсер языка, допускающий только цепочки, удовлетворяющие правилам корпоративной стандартизации, т.к при таком подходе нет возможности настройки правил по проектам. Для создания парсера языка авторами предлагается воспользоваться метасинтаксическими средствами, такими, как JavaCC, YACC, Bizon и т.д. Использование таких средств позволяет сократить время разработки парсера и быстрее адаптироваться к изменениям синтаксиса языка. Потеря в скорости анализа исходного текста, в нашем случае приемлема. На основе этого парсера создается модуль анализа корпоративных стандартов. Данный модуль должен проверить описание проекта на соответствие правилам корпоративной стандартизации, записанных в файле настройки правил корпоративной стандартизации и сохранить результаты в файле для дальнейшего использования.

5. Проектирование формата хранения результатов проверки. Результаты проверки для дальнейшей работы с ними, например, формирования требований исправления ошибок корпоративной стандартизации или вычисления показателей качества, сохраняются в файле результатов. В данном файле должны быть сохранены отношения между объектами проверки, проектом и ответственным разработчиком.
6. Проектирование базы данных для сохранения результатов проверки. Предложенный авторами XML-формат используется для обмена данными. Для анализа данных проще использовать реляционную модель, поэтому возникает необходимость в базе данных для анализа результатов и модуле, позволяющем переносить данные из файла результатов проверки в базу данных анализа.
7. Выбор и построение функций вычисления оценок, определяющих степень удовлетворения требований корпоративной стандартизации. Реализация данных оценок может быть выполнена в виде хранимых процедур в базе данных анализа.
8. Построение модуля исправления ошибок корпоративной стандартизации. Заметим, что невозможно исправить все ошибки автоматически, поэтому в данном модуле необходимо вести архив исправленных ошибок.

В качестве примера приведем часть результатов проверки проекта банковской системы на удовлетворение следующим трем правилам.

1. $\forall x \in$ Q4, где Q4 - множество идентификаторов локальных констант.

F4(x)={0, если substr(x,1,2) = ‘c_’, 1 иначе .Идентификаторы локальных констант начинаются с префикса (c_)

2. $\forall x \in$ Q6, где Q6 - множество идентификаторов локальных переменных.

F6(x)={0, если substr(x,1,2) = ‘v_’, 1 иначе . Идентификаторы локальных переменных начинаются с префикса (v_)

3. $\forall x \in$ Q8, где Q8 - множество идентификаторов локальных курсоров.

F8(x)={0 если substr(x,1,4) = ‘curs_’, 1 иначе .Идентификаторы локальных курсоров начинаются с префикса (curs_)

Пример вычисленного показателя СУМ по модулям приведен на рис 2.

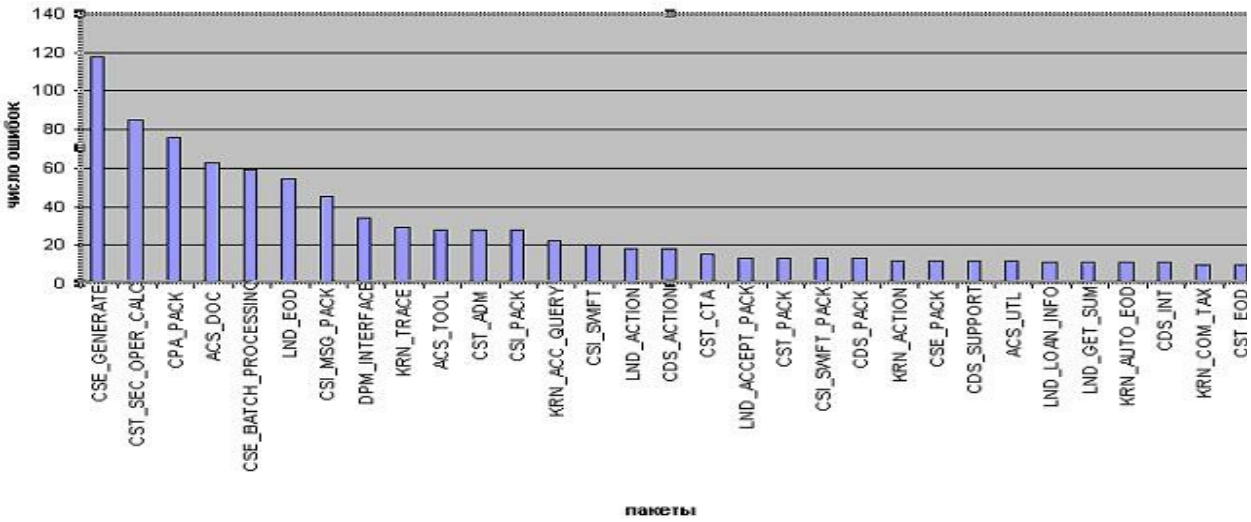

Рисунок 2.Значение показателя СУМ для одного проекта.

Пример вычисленного показателя СУМН по модулям приведен на рис 3.

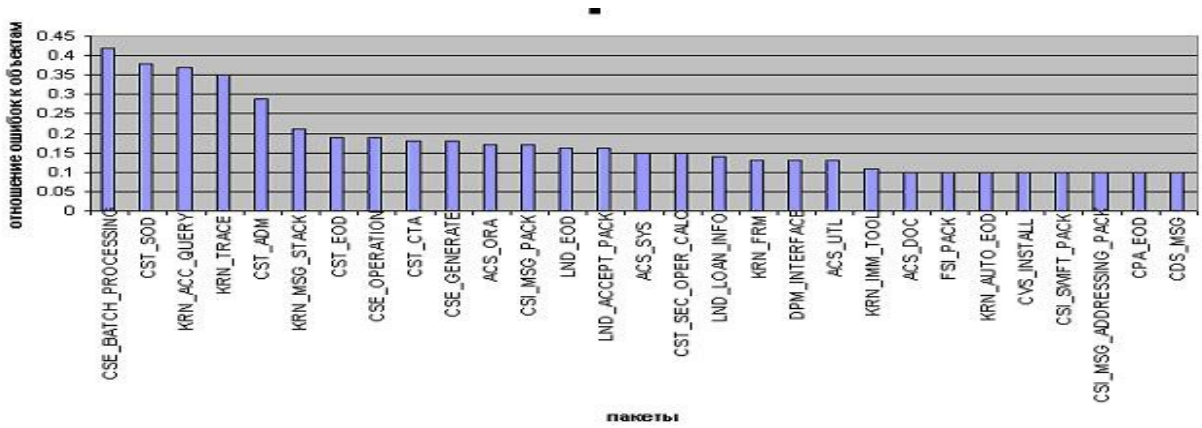

Рисунок 3.Значение показателя СУМН для одного проекта.

Анализ таких показателей позволяет выявить модули, которые требуется привести в соответствие с корпоративными стандартами и принимать управленческие решения.

Описанные авторами алгоритмы применены при разработке автоматизированной системы проверки соблюдения корпоративных стандартов для языка PL/SQL, которая в настоящий момент апробируется в компании «Форс-БС»

СПИСОК ЛИТЕРАТУРЫ

1. Ткаченко А.В. Стандарт стилевого оформления исходного кода DELPHI.- http://www.citforum.ru/programming/delphi/style_delphi/.
2. Scott W.A. Coding Standards for Java.- http://www.ambysoft.com/javaCodingStandards.html.
3. C and C++ Style Guides.- http://www.chris-lott.org/resources/cstyle/
4. Scott W.A. C++ Coding Standard.- http://www.possibility.com/Cpp/CppCodingStandard.html#init

5. Керниган Б.В. Практика программирования.- СПб.: Невский Диалект, 2001.- 380 с.
6. Терехов А.А. Промышленные методы разработки программ.- http://se.math.spbu.ru/courses/IndSE/IndSE.pdf
7. Ахо А., Ульман Дж. Теория синтаксического анализа, перевода и компиляции. - М.: Мир, 1978.- 487 с.
8. Брукс Ф. Мифический человеко-месяц или как создаются программные системы. – СПб.: Символ-Плюс, 1999.-304 с.
9. Брукшир Дж. Введение в компьютерные науки.– М.: Вильямс, 2001.-688 с.
10. Васкевич Д. Руководство по выживанию для специалистов по реорганизации бизнеса. - Киев.: Диалектика, 1996.-384 с.
11. Гласс Р., Нуазо Р. Сопровождение программного обеспечения.- М.: Мир, 1983.-156 с.
12. Барашев Д.В., Горшкова Е.А., Новиков Б.А. Оптимизация представления XML-документов в реляционной базе данных.- ru.sun.com/pdf/patterns/pattern2.pdf

СВЕДЕНИЯ ОБ АВТОРАХ

Кондаков Сергей Аркадьевич, начальник отдела разработки компании "ФОРС-БС", Москва

Кудинов Николай Александрович, аспирант кафедры математической кибернетики Московского авиационного института (государственного технического университета); E-mail: n_kudinov_@mail.ru