УДК 681.31

Автоматизированный метод корпоративной стандартизации процесса разработки систем на платформе Oracle Designer 6i.

В.И.Виноградов, Н.А. Кудинов

В статье приводится постановка задачи корпоративной стандартизации процесса кодирования и рассматривается архитектура системы контроля соблюдения требований корпоративной стандартизации для проектов с использованием продукта Oracle Designer

Проблема разработки программного обеспечения быстро, в срок и с удовлетворительным качеством стоит перед компьютерным сообществом уже давно. Еще в 70 годы Брукс в своей книге “Мифический человеко-месяц” [1] обозначил одну из основных проблем при создании программного обеспечения – нехватка времени.

Сегодня перед нами стоит та же проблема. Подавляющее большинство проектов по разработке программного обеспечения не укладываются в первоначально намеченные сроки. Обычно это приводит к удорожанию продукта и снижению его качества. Одним из используемых путей сокращения времени разработки программного обеспечения является следование стандартам качества. В последнее время наблюдается прямо-таки экспоненциальный рост числа разнообразных стандартов.

Одной из первых моделей качества стал стандарт ISO (Международной организации по стандартизации), первая версия которого была выпущена в 1987 году. С тех пор сертификаты ISO сохраняют неизменную популярность и признаются во всем мире. Международные стандарты ISO, а точнее стандарты ISO 9000, устанавливают, какие именно элементы должны включаться в систему качества, но не то, каким образом конкретная организация должна реализовать эти элементы.

Система стандартов ISO 9000 имеет ряд недостатков, устранить которые пытается, например, стандарт – Capability Maturity Model (CMM). Можно сказать, что стандарт CMM в целом состоит из критериев оценки зрелости организации и рецептов улучшения существующих процессов (указывает к чему надо стремиться). В этом наблюдается принципиальное различие с моделью, принятой в ISO, так как в ней сформулированы только необходимые условия для достижения некоторого минимального уровня организованности технологических процессов, и не дается никаких рекомендаций по дальнейшему совершенствованию процессов.

Приведенные стандарты системы качества охватывают весь процесс разработки программного обеспечения. От проектирования программного продукта и до его внедрения.

В настоящей статье рассмотрен подход, предлагаемый для повышения качества и сокращения времени, затрачиваемого на написание программного кода информационных систем (ИС) за счет стандартизации этого процесса. Такая стандартизация включает составление корпоративной системы стандартов кодирования и контроль их выполнения.

Сокращение времени, затрачиваемого на разработку ПО, достигается за счет того, что наличие стандартов разработки позволяет:

- уменьшить отвлечение ведущих разработчиков на начальной стадии работы новых разработчиков;
- избежать ряда типовых ошибок программирования;
- повысить читабельность программного кода;
- предложить программисту типовые решения ряда мелких проблем, возникающих в процессе написания программного кода, например, как назвать класс, как назвать переменную, куда ее поместить.

Рассматриваемая задача может быть формализована следующим образом.

Постановка задачи

Объектом проверки является программный компонент информационной системы, на который накладываются ограничения. Совокупность таких ограничений будем называть корпоративными стандартами.

Пусть A- множество объектов проверки: $A = \bigcup_1^n Ai$, где A_i - множество однотипных объектов проверки. P-множество ограничений, описываемых набором предикатов: для $\forall i = 1..n$ и $a \in A_i$, $P_i(a) \to \{0,1\}$

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

- разработку формальных правил кодирования или требований, предъявляемых к исходному программному коду, что выражается в составлении системы предикатов стандартизации – $\{ P_i \}$;
- алгоритмизация процедур контроля соблюдения требований корпоративных стандартов, которые выражается в вычислении функции оценки качества удовлетворения требований корпоративных стандартов кодирования;
- алгоритмизация исправления типовых ошибок нарушения требований стандартизации, что выражается в определении набора функций корректировки ошибок H: $H_i(a) = a^*$ таких, что $P_i(a^*) = 0$, $P_j(a) = 1$, где a и $a^* \in A_i$.

Система предикатов правил корпоративной стандартизации

В настоящей работе предложены следующие наборы правил, регламентирующих процесс разработки банковской информационной системы на платформе Oracle.

- Стандарт обработки ошибок на клиенте и сервере. Обеспечивает выработку единого стиля сообщений об ошибках, а также обеспечение конечного пользователя разработанных систем полной информацией о возникшей ситуации.
- Стандарт на правила формирования наименований. Описывает требования к формированию наименований таблиц, их полей, процедур, функций, переменных, констант и т. д. Улучшает читабельность и предотвращает типичные ошибки.
- Стандарт на написание клиентского и серверного кода. Описывает требования к написанию прикладными программистами исходного кода. Такие требования улучшают читабельность программ и предотвращают некоторые типовые ошибки кодирования.
- Стандарт на правила обработки транзакций. Описывает требование на использование операторов commit и rollback в PL/SQL-коде.

На основании перечисленных выше наборов правил, составлена следующая система правил, описывающих корпоративные стандарты кодирования

$\forall a \in$ A_1, где A_1 - множество зарезервированных слов языков SQL и PL/SQL.

$P_1(a)=$ { 0, если ACSII(‘a’)<=ASCII(a_i)<=ASCII(‘z’) для $\forall i$: 1<=i<=length(x),
1, в противном случае}

Все служебные зарезервированные слова SQL и PL/SQL пишутся строчными (маленькими) буквами.

$\forall a \in$ A_2, где A_2 - множество идентификаторов таблиц и полей в таблицах и курсорах.

$P_2(a)=$ { 0, если ACSII(‘A’)<=ASCII(a_i)<=ASCII(‘Z’) для $\forall i$: 1<=i<=length(x),
1, в противном случае}

Идентификаторы таблиц и полей в таблицах и курсорах пишутся прописными (заглавными) буквами.

$\forall a \in$ A_3, где A_3 - множество идентификаторов типов данных, определенных пользователем.

$P_3(a)=$ { 0, если substr(a,1,5) = ‘type_’,

1, в противном случае}

Идентификаторы типов данных, определенных пользователем, должны начинаться с префикса (type_).

$\forall a \in$ A_4, где A_4 - множество идентификаторов локальных констант.

$P_4(a)=$ { 0, если substr(a,1,2) = ‘c_’,

1, в противном случае}

Идентификаторы локальных констант должны начинаться с префикса (c_)

$\forall a \in$ A_5, где A_5 - множество идентификаторов глобальных констант.

$P_5(a)=$ { 0, если substr(a,1,3) = ‘gc_’,

1, в противном случае}

Идентификаторы глобальных констант должны начинаться с префикса (gc_).

$\forall a \in$ A_6, где A_6 - множество идентификаторов локальных переменных.

$P_6(a)=$ { 0, если substr(a,1,2) = ‘v_’,

1, в противном случае}

Идентификаторы локальных переменных должны начинаться с префикса (v_)

$\forall a \in$ A_7, где A_7 - множество идентификаторов глобальных переменных.

$P_7(a)=$ { 0, если substr(a,1,3) = ‘gv_’,

1, в противном случае}

Идентификаторы глобальных переменных должны начинаться с префикса (gc_).

$\forall a \in$ A_8, где A_8 - множество идентификаторов локальных курсоров.

$P_8(a)=$ { 0 если substr(a,1,4) = ‘curs_’,

1, в противном случае}

Идентификаторы локальных курсоров должны начинаться с префикса (curs_)

$\forall a \in$ A_9, где A_9 - множество идентификаторов глобальных курсоров.

$P_9(a)=$ { 0, если substr(a,1,5) = ‘gcurs_’,

1, в противном случае}

Идентификаторы глобальных курсоров должны начинаться с префикса (gcurs_).

$\forall a \in$ A_{10}, где A_{10} - множество идентификаторов параметров процедур и функций.

$P_{10}(a)=$ { 0, если substr(a,1,2) = ‘p_’,

1, в противном случае}

Идентификаторы параметров процедур и функций начинаются с префикса (p_).

$\forall a \in$ A_{11}, где A_{11} – множество исходных кодов процедур и функций.

$P_{11}(a)=$ { 0, если в теле программы есть вызов KRN_Trace.Start_Proc, KRN_Trace.End_Proc, Exception when others then krn_trace.Error_Handler(),

1, в противном случае}

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

$\forall a \in$ A_{12}, где A_{12} – множество исходных кодов триггеров.

$P_{12}(a)=$ { 0, если в теле программы есть вызов KRN_Trace.Start_Trigger, K KRN_Trace.Start_Trigger,

1, в противном случае}

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.

$\forall a \in$ A_{13}, где A_{13} – множество исходных кодов процедур, функций или триггеров в которых используется оператор return .

$P_{13}(a)$={ 0, если в теле программы есть вызов KRN_Trace.End_Proc или KRN_Trace.End_Trigger перед оператором return,

1, в противном случае}

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.

$\forall a \in$ A_{14}, где A_{14} – множество исходных кодов курсоров.

$P_{14}(a)$={ 0, если в теле программы не используются переменные ,

1, в противном случае}

При объявлении курсоров не допускается использовать определенные вне их тела переменные - передача значений осуществляется только через их параметры (аргументы).

$\forall a \in$ A_{15}, где A_{15} – множество исходных кодов процедур и функций.

$P_{15}(a)$={ 0, если в теле программы не используются переменные ,

1, в противном случае}

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).

$\forall a \in$ A_{16}, где A_{16} – множество исходных кодов процедур и функций и триггеров.

$P_{16}(a)$={ 0, если в теле программы нет оператора select .. into ,

1, в противном случае}

В триггерах, пакетах, процедурах и функциях все операторы SELECT желательно выполнять в виде курсоров

Система контроля исходного кода на соответствие стандартам

Разработать правила стандартизации - это лишь часть задачи стандартизации разработки. Даже добросовестные специалисты нуждаются в средствах контроля своей работы, в качестве которых может выступать как человек (контролер, тестировщик), так и некоторая автоматизированная процедура. Методика проверки соблюдения правил стандартизации в проекте может заключаться в визуальном контроле определенными людьми результатов работы программистов [4] или в автоматизированном контроле.

В частности, при визуальной проверке можно произвести оценку экранной формы на соответствие корпоративным стандартам. С программным кодом дела обстоят сложнее.

Кроме людей занимающихся непосредственно разработкой обычно его никто не смотрит. И поэтому необходимы процедуры его автоматической проверки.

Для этих целей предлагается разработать специализированное программное обеспечение, которое будет осуществлять проверку программного кода на соответствие стандартам кодирования и стандартам наименования.

Архитектура такого программного комплекса представлена на рисунке 1.

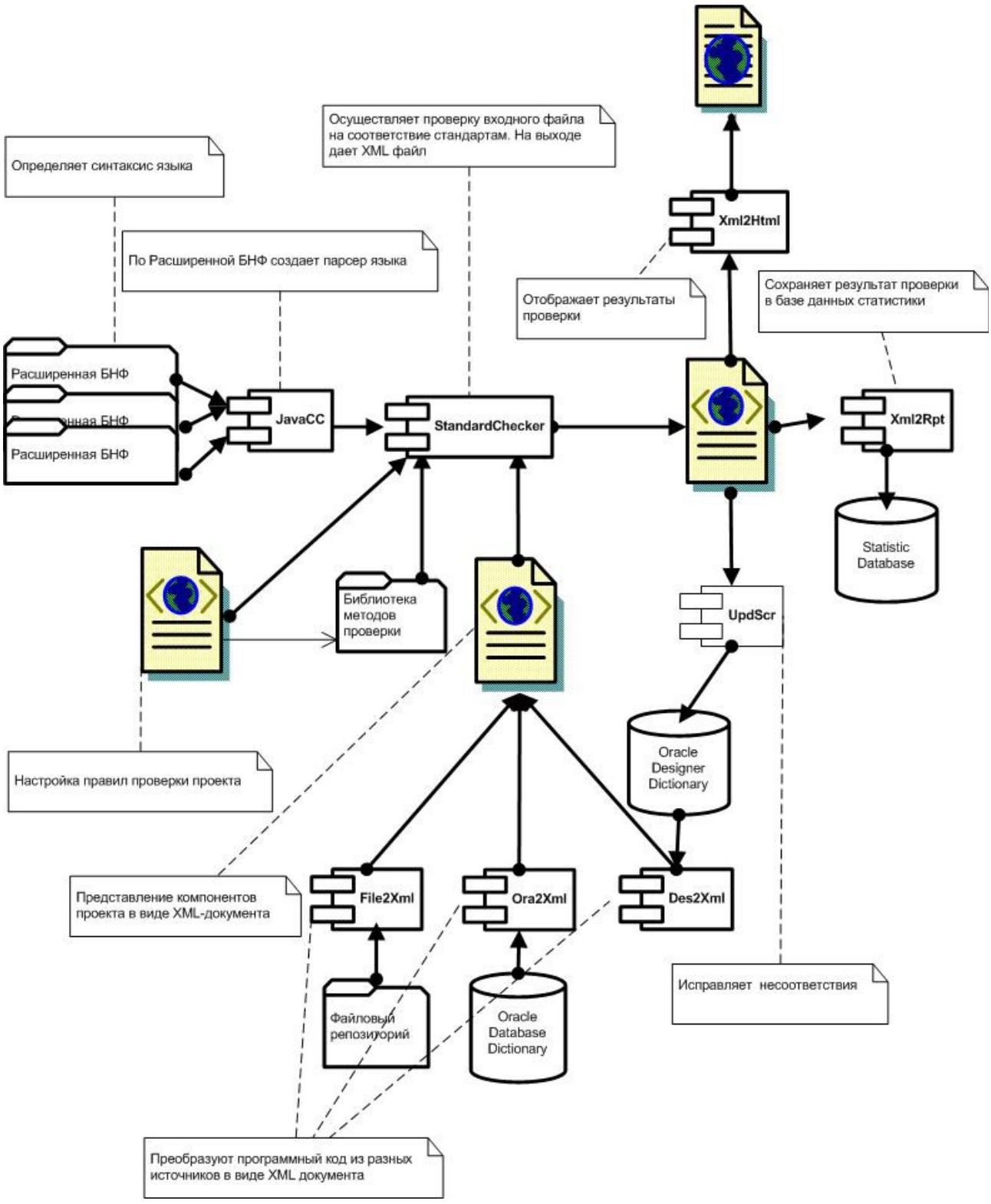

Рисунок 1. Архитектура системы контроля соблюдения требований стандартизации

Составными частями этой архитектуры являются следующие элементы.

Расширенная БНФ – грамматика языка программирования, для которого вводится система стандартов кодирования. Грамматика языка записывается в расширенной форме Бэкуса-Наура.

JavaCC - генератор синтаксических анализаторов. JavaCC обеспечивает расширение языка Java, для задания грамматики нового языка программирования или для порождения синтаксических анализаторов других языков.

Классы проверки - множество классов, написанных на языке Java, реализующих правила проверки.

XML-описание проекта – документ, в формате XML, содержащий структурированное описание проекта.

StandardChecker – класс, реализующий проверку входящего XML–описания проекта на соответствие корпоративным стандартам кодирования.

Файловый репозиторий – множество каталогов проекта, содержащих объекты проекта, хранящиеся в формате файловой системы.

File2Xml – преобразователь файлов проекта из файлового репозитория в специализированное XML-описание проекта для возможности дальнейшей проверки.

Oracle Database Dictionary – словарь базы данных Oracle, содержащий описание проекта во внутренем формате Oracle Database.

Ora2Xml - преобразователь из словаря Oracle Database в специализированное XML-описание проекта для возможности дальнейшей проверки.

Oracle Designer Dictionary – репозиторий средства разработки Oracle Designer, в котором хранится описание проекта в формате Oracle Designer.

Des2Xml - преобразователь описания проекта из репозитория Oracle Designer в специализированное XML-описание проекта для возможности дальнейшей проверки

UpdScr – компонет исправления исходных кодов проекта, приводящий их в соответствие с корпоративными стандартами.

Xml2Html – компонент преобразования файла результата проверки в HTML формат.

Xml2Rpt – компонент занесения результатов проверки проекта в базу данных результатов.

Схемы контроля

Проверка исходного кода осуществляется по одной из следующих схем, приведенных на рисунке 2.

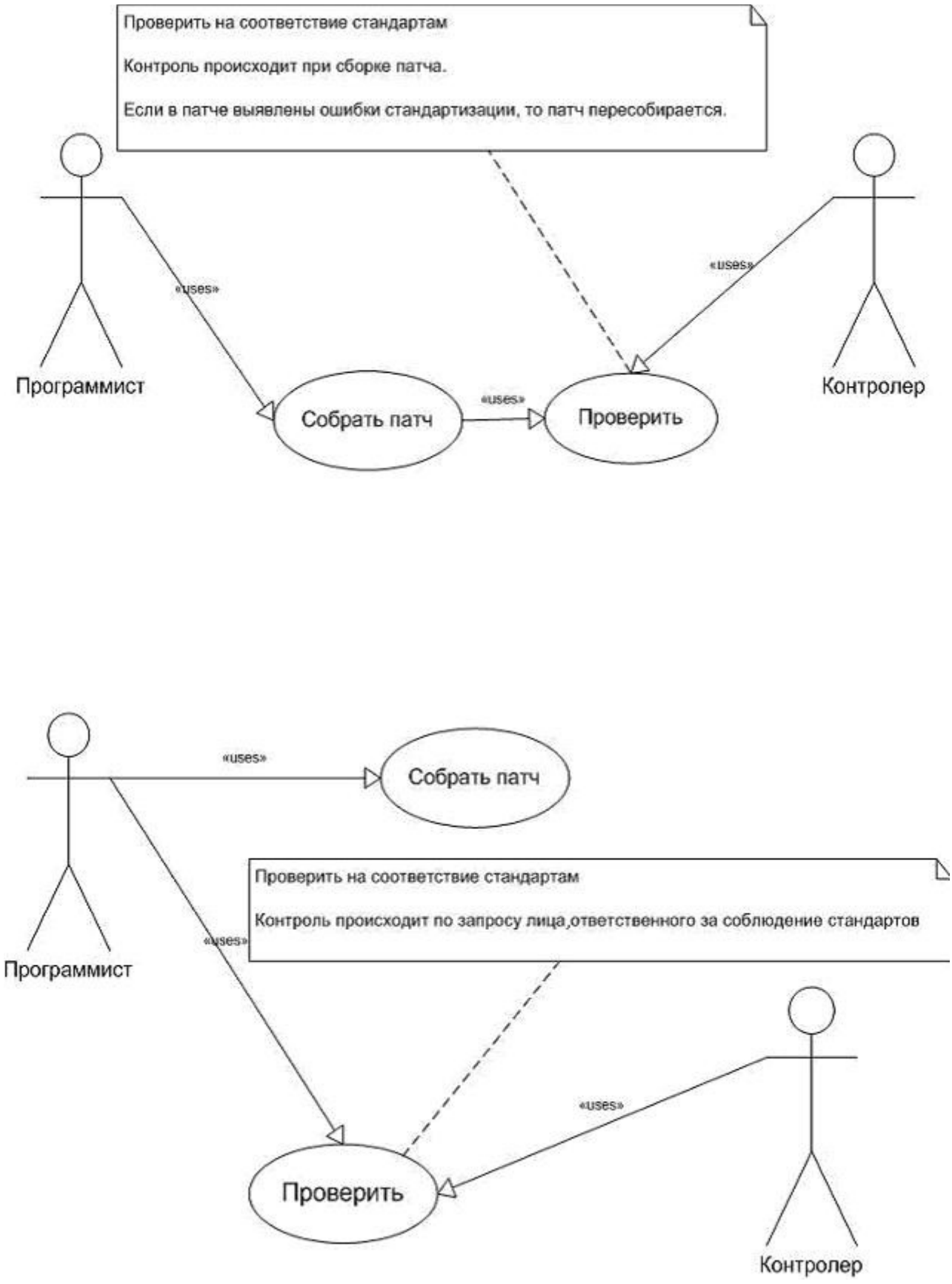

Рисунок 2. Схемы проверки соответствия стандартам

Наличие двух схем проверки отражает тот факт, что первоочередной целью софтверной компании является выпуск программного продукта. И далеко не всегда необходимо, чтобы продукт, выпускаемый компанией, был выполнен без ошибок. Часто бывает так, что компания принимает решение выпустить продукт, не соответствующий стандартам, чтобы «удержаться на плаву» или обойти конкурентов.

Но есть также ситуации, когда качество играет исключительную роль. Для этого существует схема контроля при сборке патча.

Архитектура модуля проверки исходного кода

Диаграмма классов модуля проверки исходного кода изображена на рисунке 3.

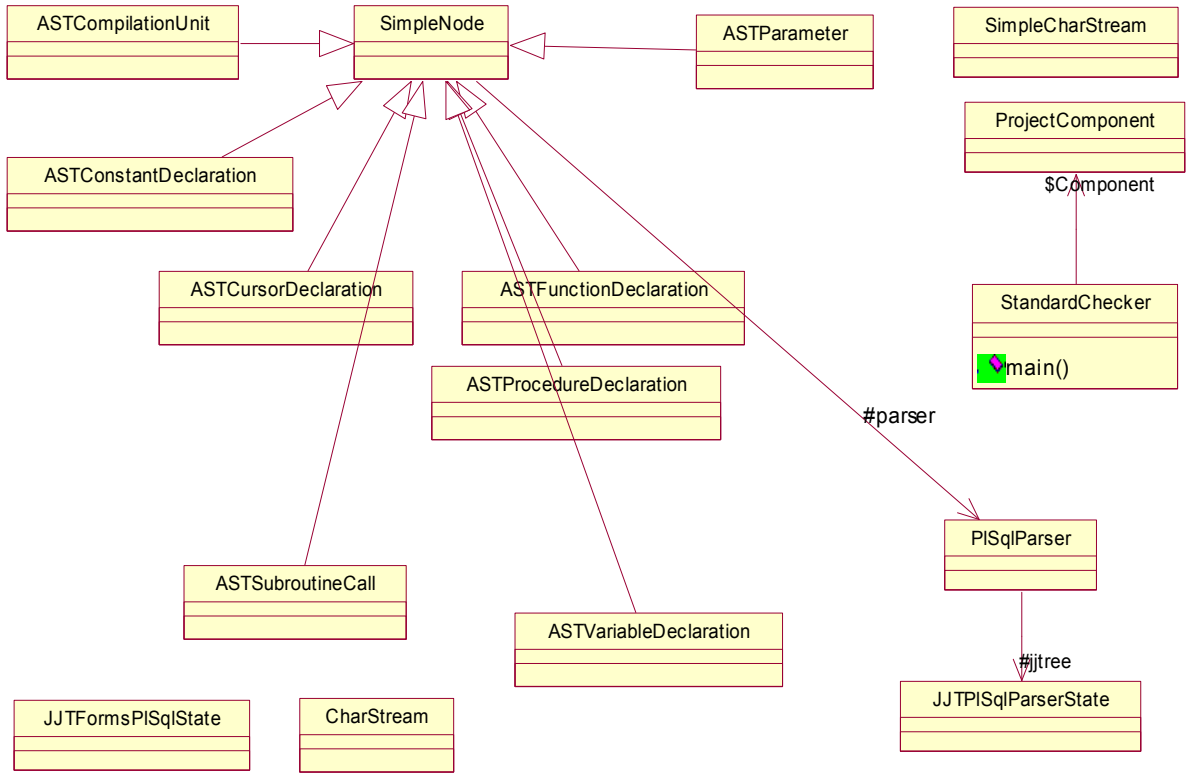

Рисунок 3. Диаграмма классов модуля проверки исходного кода

Класс StandardChecker

Назначение: Используется для проверки специализированного XML-описания проекта (наименование которого получает из командной строки) на соответствие стандартам.

Методы:

Main - проверка входного файла на соответствие стандартам.

Класс ProjectComponent

Назначение: Используется для считывания описания компонента проекта из специализированного XML-описания. Предоставляет методы проверки компонента.

Класс CharStream

Назначение: Используется для абстрагирования от того, как передан код модуля. Предоставляет интерфейс для работы с кодом модуля.

Класс PLSQLParser

Назначение: Строит дерево синтаксического разбора для потока символов.

Класс SimpleNode

Назначение: Используется для представления объекта проверки - элемента программы.

Поля:

Name – наименования узла

ErrorDescription – Текст сообщения об ошибке несоответствия стандартам, если такая имеется.

Методы:

Класс определяет метод CheckCurrentNode(), который переопределяется в его потомках. Метод вызывается при обходе дерева. Служит для проверки узла на соответствие стандартам.

Класс ASTParameter

Назначение: Используется для представления параметра процедуры/функции программы на PL/SQL при синтаксическом разборе.

Класс ASTConstantDeclaration

Назначение: Используется для представления объявления константы программы на PL/SQL при синтаксическом разборе.

Класс ASTVariableDeclaration

Назначение: Используется для представления переменной программы на PL/SQL при синтаксическом разборе.

Класс ASTFunctionDeclaration

Назначение: Используется для представления объявления функции программы на PL/SQL при синтаксическом разборе.

Класс ASTProcedureDeclaration

Назначение: Используется для представления объявления процедуры программы на PL/SQL при синтаксическом разборе.

Класс ASTPackageDeclaration

Назначение: Используется для представления объявления пакета программы на PL/SQL при синтаксическом разборе.

Класс ASTCursorDeclaration

Назначение: Используется для представления объявления курсора программы на PL/SQL при синтаксическом разборе.

Модуль работы с результатами проверки

При использовании системы контроля соответствия стандартам важной задачей является возможность наблюдения за динамикой ошибок стандартизации. То есть возможность получения информации о количестве различного вида ошибок и его изменение в различных версиях программного продукта. На следующем рисунке приведена упрощенная ER-модель состава данных модуля работы с результатами проверок.

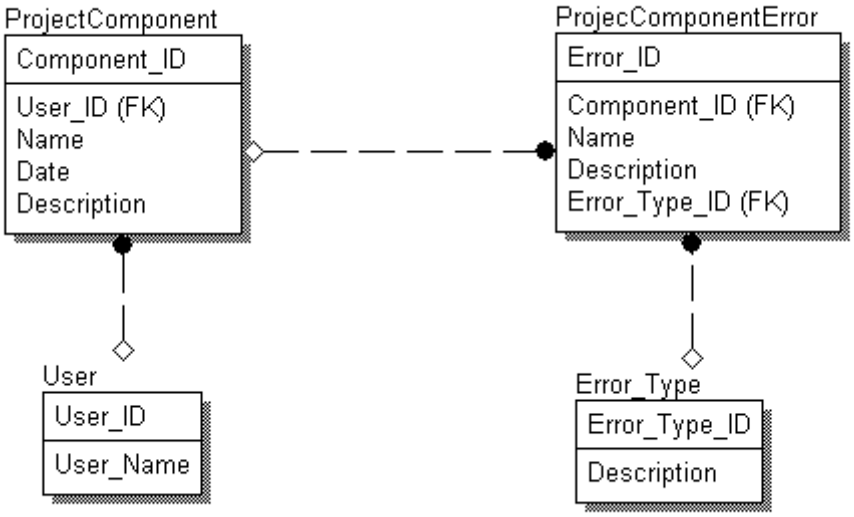

Рисунок 4. ER-Диаграмма модуля результатов проверок ошибок стандартизации

Модуль предназначен, в первую очередь, для руководителей проектов и позволяет контролировать соблюдение правил корпоративных стандартов, вычислять метрики качества удовлетворения корпоративных стандартов.

Заключение

Разработанная система позволяет контролировать нарушения правил корпоративной стандартизации, а также производить их частичное исправление. Для руководителя проекта

предназначен модуль контроля за динамикой появления/исправления ошибок стандартизации.

СПИСОК ЛИТЕРАТУРЫ

1. Фредерик Брукс. Мифический человеко-месяц. - СПб.: Символ, 2000.-304 с.
2. А. Ахо, Дж. Ульман. Теория синтаксического анализа, перевода и компиляции - М.: Мир, 1978.- 487 с.
3. Б.В.Керниган. Практика программирования. СПб.: Невский Диалект 2001.- 380с.
4. К.Бек Экстремальное программирование. СПб.: Питер, 2002.- 190 с.

СВЕДЕНИЯ ОБ АВТОРАХ

Виноградов Владимир Иванович, доцент кафедры математической кибернетики Московского авиационного института (государственного технического университета), к.ф.-м.н.

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