УДК 004.3.068; 004.424

Е. Г. Князев, Д. Г. Шопырин, канд. техн. наук,
Санкт-Петербургский государственный
университет информационных технологий,
механики и оптики

Автоматизированная классификация изменений программного кода методами многомерного статистического анализа

Одной из основных мер общего контроля состояния программной системы является постоянный обзор изменений исходного кода. Однако эта деятельность связана с анализом большого потока информации. Представлен метод автоматизированной классификации изменений программного кода, основанный на многомерном статистическом анализе. Автоматизированная классификация изменения программного кода позволяет увеличить производительность эксперта при выполнении задач, связанных с контролем состояния и анализом истории программных систем.

1. Постановка задачи

При промышленной разработке программного обеспечения часто вычисляются группы разработчиков с явным техническим лидером, например, старшим или ведущим программистом. В обязанности технического лидера обычно входит не только общий контроль состояния программной системы, но и обучение менее опытных разработчиков на примере ошибок, которые они совершают в процессе разработки. Одной из основных мер общего контроля состояния программной системы является постоянный обзор изменений исходного кода. Однако технический лидер зачастую сталкивается с очень большим потоком изменений, который зависит от размеров команды разработчиков и используемого цикла разработки. Например, в случае команды из пяти разработчиков и использования современных *гибких* (*agile*) процессов разработки [1] можно ожидать, что техническому лидеру придется просматривать как минимум 25 изменений ежедневно.

Автоматизированная классификация изменений, в качестве вспомогательного инструмента, позволяет увеличить производительность эксперта при выполнении задач, связанных с анализом истории программных систем. В частности, использование автоматизированной классификации позволяет отфильтровать несущественные для эксперта изменения системы. Например, при анализе текущего состояния проекта эксперт может выделить изменения, которые привели к реализации новой функциональности и сосредоточиться на них.

Также автоматизированная классификация изменений может применяться для следующих задач:

- *обеспечение возможности запрета изменений на определенных этапах разработки.* К примеру, на этапе стабилизации программного кода продукта может быть автоматизирована проверка запрета реализации новой функциональности;
- *контроль процесса разработки программного обеспечения с помощью анализа соотношений вносимых изменений.* Автоматизированная классификация изменений помогает оценивать скорость и качество разработки программного продукта по количественному соотношению их типов. Так, преобладание исправлений ошибок на стадии реализации новой функциональности будет свидетельствовать о необходимости введения стадии стабилизации вместо продолжения фазы развития программного продукта.

В настоящей статье предлагается метод классификации изменений программного кода, который позволяет автоматически проводить разделение семантически различных изменений на основе значений метрик исходного кода.

В разделе 2 данной статьи приводится краткий обзор методов *MSR* (*Mining Software Repositories, анализ исторических хранилищ программного кода*) [2], позволяющих решать задачи классификации изменений путем анализа изменений исходного кода. В разделе 3 описывается предложенный метод классификации изменений. В разделе 4 рассматривается пример практического применения инструмента классификации изменений при разработке программного продукта *Navi-Manager* в компании "*Транзас Технологии*".

2. Предшествующие исследования

Среди существующих методов классификации изменений можно выделить следующие группы [3]:

- *неформальные методы* — такие, как автоматизированная классификация изменений посредством анализа комментариев [4, 5];
- *методы анализа синтаксиса изменений* — такие, как эвристическое сравнение синтаксических деревьев версии [6] и анализ разницы версии с помощью встраиваемых в исходный код тегов [7].
- *методы, основанные на data mining* — такие, как метод кластеризации исходного кода на основе метрик [8].

Неформальные методы. В работе [4] предложен метод автоматизированной классификации, основанный на анализе комментариев к изменениям кода. В комментариях осуществляется поиск слов, специфичных для каждого из типов изменений. Если вхождение слова найдено, изменение классифицируется как принадлежащее группе, описываемой этим словом. Например, слова "*ис-*

правлено", "*ошибка*" характеризуют группу исправлений ошибок, а слова "*добавлено*", "*реализовано*" — группу добавления новой функциональности. Далее сравниваются результаты автоматизированной и экспертной оценок для тестового набора данных. При таком подходе согласованность результатов достигает 70 %.

Недостатком данного метода является то, что он основывается на анализе комментариев. Однако разработчики не всегда исчерпывающе и корректно описывают проведенные ими модификации кода. Вообще, содержание комментария субъективно, и текст, написанный человеком, может быть интерпретирован различным образом. Поэтому на практике не всегда корректно судить о содержании изменений, основываясь лишь на комментариях к ним.

Методы анализа синтаксиса изменений. В работе [6] с помощью эвристического алгоритма сравнения синтаксических деревьев, построенных для последовательных версий исходных файлов, строится синтаксическая разница между версиями. Затем вычисляется, сколько функций и вызовов было добавлено, модифицировано, сколько условных выражений было изменено и т. д.

В работах [7, 9] анализируется разница версий с помощью тегов, предварительно встроенных в код. Благодаря тегам обнаруживаются добавленные, измененные и удаленные синтаксические вхождения и вычисляются типы синтаксических изменений. В результате работы метода даются ответы на вопросы: добавились ли новые методы в определенный класс, есть ли изменения в директивах препроцессора, было ли модифицировано условие любого условного оператора и т. д.

Недостатками методов анализа синтаксиса изменений является то, что они в большинстве своем сложны и зависят от конкретного языка программирования. Применение таких методов оправдано только для анализа простых изменений. В случае, если структура изменения сложна, что часто встречается на практике, метод не позволит дать однозначного ответа на вопрос о его типе.

Методы, основанные на data mining. На данный момент авторам неизвестны методы автоматизированной классификации изменений программного кода, основанные *data mining* (*в дословном переводе — "добыча данных"*). Однако многомерный статистический анализ используется для других смежных задач. Например, в работе [8] предложен метод анализа архитектуры программной системы на основе кластеризации ее компонентов связности. Для множества подпрограмм исходного кода вычисляются метрики связности. Затем значения этих метрик кластеризуются. Каждый кластер представляет собой отдельный модуль, составленный из сильно связанных компонентов, в то время как сам модуль слабо связан с остальными модулями. В результате, с помощью оценивания числа кластеров в системе можно определить степень ее модульности. Модульность, в свою очередь, — это показатель качества системы.

Принципиальным отличием предложенного в настоящей работе метода является применение анализа *изменений* исходного кода, а не только текущего состояния системы, как в работе [8]. Это позволяет, распространив анализ кода во временную плоскость, исследовать *процесс развития* программ. Предложенный метод дает возможность решать задачу автоматизированной классификации изменений программного кода без применения синтаксического анализа. Также предложенный метод позволяет выявлять скрытые зависимости, которые не могут быть обнаружены простым анализом исходного кода.

3. Метод кластеризации метрик изменений

Метод автоматизированной классификации изменений, предложенный в настоящей работе, относится к *методам data mining*. Он основан на кластеризации значений метрик изменений исходного кода с помощью метода *Мак-Куина* [10, 11]. В результате проводится разбиение множества изменений на заданное число кластеров, каждый из которых соответствует определенному типу изменений.

Формализация задачи. *Изменение кода программной системы* δ трактуется в работе как отображение множества исходных данных S в другое множество, модифицированных данных S^*:

$$\delta : S \xrightarrow{\delta} S^*.$$

В некоторых современных системах хранения исходного кода каждому состоянию исходного кода последовательно сопоставляется неотрицательное целое число r, которое называется *ревизией* или *версией программного кода*. Поэтому каждое изменение исходных данных можно описать следующим образом:

$$\delta_r : S_r \xrightarrow{\delta_r} S_{r+1}.$$

Каждое изменение δ_r может быть отнесено экспертом к некоторому множеству типов изменений t_i, где $t_i \in T$ — тип изменения δ_r. При этом T представляет собой множество типов изменений, специфичное для каждого конкретного проекта. Состав множества T определяется экспертом в зависимости от специфики проекта. Во множество T обычно входят такие типы изменений, как *реализация новой функциональности, рефакторинг, исправление ошибки* и т. д. [12].

Задача отнесения изменения к тому или иному типу изменений трудоемка и требует высокой квалификации эксперта, так как нет четких критериев оценки типа изменения. Введем функцию интерпретации изменений I, отображающую множество изменений $\{\delta_r\}$ во множество их типов $\{t_i\}$:

$$I : \{\delta_r\} \xrightarrow{I} \{t_1, t_2, \ldots, t_n\}.$$

Предлагается автоматизировать процесс выделения типов изменений с помощью кластеризации метрик изменений. В процессе кластеризации строится множество кластеров изменений C такое, что каждое изменение δ_r относится к некоторому кластеру $c_j \in C$.

Введем функцию автоматизированной классификации изменений I_A, отображающую множество изменений $\{\delta_r\}$ во множество их типов $\{t_i\}$:

$$I_A : \{\delta_r\} \xrightarrow{I_A} \{t_1, t_2, ..., t_n\}.$$

Здесь функция автоматизированной классификации I_A есть композиция функций кластеризации I_C и интерпретации кластеров I_T:

$$I_A = I_C \circ I_T;$$

$$I_C : \{\delta_r\} \xrightarrow{I_C} \{c_1, c_2, ..., c_m\};$$

$$I_T : \{c_1, c_2, ..., c_m\} \xrightarrow{I_T} \{t_1, t_2, ..., t_n\}.$$

Функция кластеризации I_C отображает множество изменений во множество кластеров. Функция интерпретации кластеров I_T отображает множество кластеров C в множество типов изменений T.

Функция кластеризации I_C может быть построена с помощью метода многомерной кластеризации Мак-Куина [10, 11]. Кластеризацию изменений будем осуществлять на основе некоторых метрик изменений $M'\delta_r$. Определим понятие метрики изменения через понятие метрики программного обеспечения.

Метрика программного обеспечения (*software metric*) — это мера M, позволяющая получить числовое значение некоторого свойства программного обеспечения S или его спецификаций [13, 14], например, число строк исходного файла, цикломатическая сложность [15], число ошибок на строку кода, число классов и интерфейсов, связность и др.

Тогда *метрику изменения программного обеспечения* можно определить как разность значений метрики измененного кода MS_{r+1} и метрики исходного кода MS_r:

$$M'\delta_r = MS_{r+1} - MS_r. \qquad (1)$$

Выберем набор метрик программного обеспечения $\overline{M} = <M_1, M_2, ..., M_p>$. Тогда для каждого изменения δ_r можно построить набор метрик изменения:

$$\overline{M}'\delta_r = < M_1'\delta_r, M_2'\delta_r, ..., M_p' \delta_r>.$$

Тогда $\overline{M}'\delta_r$ — это точка в k-мерном пространстве кластеризации. Мерой расстояния между точками в этом пространстве выберем евклидово расстояние ρ:

$$\rho(<x_1, x_2, ..., x_k>, <y_1, y_2, ..., y_k>) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_k - y_k)^2}.$$

Теперь разбиение на кластеры может быть задано следующим образом:

$$C = \{c_1, c_2, ..., c_m\}, c_j = \{\delta_r, \delta_g | \rho(\overline{M}'\delta_r, \overline{M}'\delta_g) < \rho_{\min}\}, \qquad (2)$$

где $\rho_{\min}$ — величина, определяющая меру близости для включения объектов в один кластер.

Алгоритм кластеризации изменений. Пусть известно число кластеров m, выбран набор метрик $\overline{M}$ и мера расстояния ρ между точками пространства кластеризации принята евклидовой. В соответствии с методом многомерной кластеризации Мак-Куина [10, 11], алгоритм кластеризации изменений следующий:

1. Выполнить начальное разбиение множества объектов $\{\delta_r\}$ случайным образом:

$$C^0 = \{c_1, c_2, ..., c_n\}, c_i = \{\delta_r | \delta_r \notin c_j, j \neq i\}.$$

2. Принять номер итерации $l = 1$.

3. Определить центры кластеров cc_i по формуле

$$cc_i = \frac{\sum_r [\delta_r \in c_i] \overline{M}'\delta_r}{\sum_r [\delta_r \in c_i]}.$$

4. Обновить множества распределения объектов по кластерам $C^l = \{c_i\}$:

$$c_i = \{\delta_r | \rho(\overline{M}'\delta_r, cc_i) = \min \rho(\overline{M}'\delta_r, cc_j)\}.$$

5. Проверить условие $\sum_i \| c_i^l \Delta c_i^{l-1} \| = 0$, где Δ — операция взятия симметрической разности множеств: $A \Delta B = (A \cup B) \backslash (A \cap B)$. Если условия выполнены, то завершить процесс, иначе перейти к шагу 3 с номером итерации $l = l + 1$.

Приведенный алгоритм позволяет автоматически разбить множество изменений на кластеры. В каждый кластер группируются наиболее схожие друг с другом изменения.

Интерпретация результатов кластеризации. Подбор подходящего числа кластеров m и построение функции интерпретации кластеров I_T проводится экспертом на основе выборочного анализа изменений, принадлежащих каждому кластеру. Эти задачи значительно менее трудоемки, чем исходная задача, так как на практике имеет смысл различать лишь небольшое число типов изменений.

В процессе построения функции интерпретации кластеров I_T экспертом анализируются изменение исходного кода и комментарий, сопровождающий изменение. В результате устанавливается, какому из типов t_j соответствует данный кластер c_i. При невозможности сопоставления кластера изменений экспертному типу следует повторно обратиться к выбору метрик для кластеризации.

Таблица 1

Метрики, используемые в процессе кластеризации

Метрика	Описание
eLOC	Эффективное число строк кода
CC	Цикломатическая сложность (число независимых путей в графе исполнения)
IC	Интерфейсная сложность (общее число параметров во всех методах)
C/S	Число классов и структур

4. Пример практического применения метода автоматизированной классификации

Представленный в работе метод успешно опробован на практике. Реализован программный инструмент кластеризации ревизий репозитория системы контроля версий *Subversion*. Метрики исходного кода, написанного на любом из языков C, C++, C#, *Java*, вычисляются с помощью инструмента *MSquared Resource Standard Metrics* [17]. Разработанный инструмент внедрен в компании "*Транзас Технологии*" и доказал эффективность практического применения предложенного метода.

В качестве иллюстрации метода кластеризации метрик изменений в текущем разделе приводятся результаты применения метода на проекте *Navi-Manager*. *Navi-Manager* — это система слежения за флотом, состоящая из серверной и клиентской частей. Система преимущественно реализована на языке C#. Общее число строк кода — более 100 000.

В качестве исходных данных анализа случайным образом выбрано 13 изменений, выполненных в проекте *Navi-Manager* за шесть месяцев разработки, и проведена экспертная классификация этих изменений по типам. Экспертом выделены следующие типы изменений в наборе: добавление новой функциональности, удаление функциональности, исправление ошибки, рефакторинг, косметические изменения.

Первоначальный выбор набора метрик для автоматизированной классификации осуществляется на основе эмпирических представлений эксперта о способности разделения экспертных типов изменений алгоритмом кластеризации по различиям значений их метрик. Выбранные метрики, предположительно связанные с различением экспертных типов изменений, приведены в табл. 1 [0].

В процессе автоматизированной кластеризации было построено множество из четырех кластеров, которые с помощью эксперта были сопоставлены выделенным типам изменений. Результаты этого сопоставления приведены в табл. 2.

В табл. 2 приведены:

- ревизия изменения;
- комментарий к изменению;
- тип изменения, который был присвоен изменению в процессе оценки экспертом;
- кластер, который был сопоставлен изменению в процессе кластеризации;
- интерпретация кластеров экспертом.

В табл. 2, 3 использованы следующие сокращения:

- "Удал." — удаление кода;
- "Реф." — рефакторинг;
- "Испр." — исправление ошибки;
- "Нов." — новая функциональность.

Для оценки согласованности автоматизированной и экспертной классификации в работе используется *коэффициент Кохена* [20, 21]. Коэффициент Кохена представляет собой меру согласия, с которой два эксперта конкурируют в своих сортировках N элементов по k взаимно исключающим категориям. Эксперта в данном контексте может представлять человек или множество людей, которые коллективно распределяют N элементов, или некоторый алгоритм, который распределяет элементы на основе некоторого критерия.

Коэффициент Кохена вычисляется следующим образом. В табл. 3 размером $(n + 1)x(n + 1)$ выписываются результаты экспертной и автома-

Таблица 2

Результаты автоматизированной и экспертоной классификации изменений из проекта *Navi-Manager*

Ревизия	Комментарий	Тип изменений	Номер кластера	Интерпретация
12883	Перенос файлов между проектами, удаление ненужных	Удал.	1	Удал.
18281	Уменьшено дублирование кода	Удал.		
17135	Обобщена логика FindOrCreateUserSource	Реф.	2	Реф.
18048	Рефакторинг работы с интервалами	Реф.		
16743	Обобщена обработка GlobeWireless	Реф.		
17613	Увеличен таймаут PositionDiscarder до 5 мин	Испр.	3	Испр.
17929	Исправлено несколько логических ошибок в работе TrackAgent	Испр.		
17970	Переименован ITrackFetch.Interval в DestInterval	Реф.		
18273	Поправлена ошибка при проверке флагов рисования элемента меню	Испр.		
15115	Поправлена установка vessel.LastReportTime	Испр.	4	Нов.
15191	Добавлены блоки try..catch в методы OnStart, OnStop сервиса	Нов.		
16580	Добавлен перенос max LastReportTime, LastSourceId в логику объединения данных объекта	Нов.		
16899	Добавлена инверсия цвета текста в колонку L в Vessel List View	Нов.		

Таблица 3

Соответствие автоматизированной и экспертной классификации измерений *Navi-Manager*

Экспертная классификация	Автоматизированная классификация				
	Удал.	Реф.	Испр.	Нов.	Сумма
Удал.	2				2
Реф.		3	1		4
Испр.			3	1	4
Нов.				3	3
Сумма	2	3	4	4	13

тизированной классификации (число кластеров должно быть равно числу экспертных типов).

Элемент табл. 3 p_{ij} — число изменений, принадлежащих к типу t_i автоматизированной классификации и одновременно к типу t_j экспертной классификации. На главной диагонали расположены величины, равные числу изменений, для которых совпадает результат автоматизированной и экспертной классификации. В строке "*Сумма*" приведено распределение частот изменении по типам автоматизированной классификации. В столбце "*Сумма*" приведено распределение частот изменений по экспертным типам. Правый нижний элемент табл. 3 содержит общее число анализируемых изменений.

Выражение для расчета коэффициента согласия Кохена следующее:

$$\kappa = \frac{p_\alpha - p_e}{1 - p_e},$$

где p_α — относительное наблюдаемое согласие между экспертами; p_e — вероятность обусловленности этого согласия случайностью. Если эксперты находятся в абсолютном согласии между собой, тогда $\kappa = 1$. Если же согласие между экспертами отсутствует (но не по причине случайности), тогда $\kappa \leqslant 1$.

Вычисление коэффициента согласованности автоматизированной и экспертной классификации выполняются по кросс-табуляции наблюдаемого числа наблюдений изменений по каждому типу (табл. 3). Тогда величина p_α рассчитывается как отношение суммы элементов табл. 3 по диагонали к общему числу элементов:

$$p_\alpha = \frac{\sum_i p_{ii}}{\Sigma}.$$

Величина p_e вычисляется как отношение суммы диагональных элементов таблицы, полученной случайным распределением по категориям, к общему числу элементов:

$$p_e = \sum_k p_{k+} p_{+k},$$

где $p_{k+} = \frac{\sum_j p_{kj}}{\Sigma}$ и $p_{+k} = \frac{\sum_i p_{ik}}{\Sigma}$ — пропорции элементов по строкам и колонкам от общего их числа.

По результатам эксперимента получено значение каппа, равное 0,79, которое лежит на границе значительной и превосходной степени согласованности двух методов классификации [21].

Заключение

Преимущества предложенного метода состоят в следующем:

- *объективность:* для анализа используется исходный код, а не, например, комментарий, сопровождающий изменение. Оценка по исходному коду адекватна в отличие от классификации комментариев к изменениям, ведь комментарии могут не в полной мере соответствовать характеру изменений [22];
- *настраиваемость:* множество метрик программного кода может выбираться в зависимости от того, по каким аспектам изменений предполагается группировка [13, 14];
- *адаптивность:* при кластеризации задается лишь результирующее число групп. Следовательно, для каждого отдельно взятого проекта предложенный метод позволяет выделить специфичные множества изменений, которые затем интерпретируются как те или иные семантические группы изменений;
- *формальность:* кластеризация осуществляется с помощью формальных методов добычи данных вообще и анализа программных репозиториев, в частности.

Практическое применение метода выявило ряд еще не решенных проблем. Например, возникает проблема выбора числа групп для кластеризации. Эта проблема описана, например, в работе [11]. Задача определения числа кластеров решается опытным путем, а именно, выполнением алгоритма кластеризации над исходными данными с несколькими значениями *n*. Дополнительно предполагается участие эксперта в процессе интерпретации результатов автоматической кластеризации.

Также стоит проблема *смешанных изменений*, сочетающих в себе разнородные модификации кода. Настоящим методом не всегда возможна корректная классификация таких изменений. Нужно заметить, что наличие смешанных изменений на практике нежелательно и даже вредно. При их наличии усложняется процедура просмотра кода и другая работа с историей программного продукта. Выделение смешанных изменений в отдельный тип в процессе кластеризации — предмет дальнейших исследований.

Решение этих и других проблем является целью дальнейших исследований. Однако уже сей-

час, в результате анализа качества автоматизированной классификации для тестовых выборок, делается вывод о практической способности предложенного метода классифицировать изменения программного кода адекватным образом.

Список литературы

1. **Aoyama M.** Agile software process and its experience // IEEE Computer Society. Proceedings of the 20th international Conference on Software Engineering (Kyoto, Japan, April 19—25, 1998). International Conference on Software Engineering. Washington: DC. 1998. P. 3—12.
2. **Zimmermann T.** Knowledge Collaboration by Mining Software Repositories // Proceedings of the 2nd International Workshop on Supporting Knowledge Collaboration in Software Development (KCSD 2006). Tokyo, Japan, 2006. P. 64, 65.
3. **Kagdi H., Collard M., Maletic J.** Towards a Taxonomy of Approaches for Mining of Source Code Repositories // ACM SIGSOFT Software Engineering Notes, Proceedings of the 2005 international workshop on Mining software repositories MSR '05. St. Louis, Missouri. 2005. P. 1—5.
4. **Hassan A. E., Holt R. C.** Source Control Change Messages: How Are They Used And What Do They Mean? 2004. http://www.ece.uvic.ca/~ahmed/home/pubs/CVSSurvey.pdf.
5. **Mockus A., Votta L. G.** Identifying reasons for software change using historic databases // Proceedings of the International Conference on Software Maintenance (ICSM). San Jose, California. 2000. P. 120— 130.
6. **Raghavan S., Rohana R., Podgurski A., Augustine V.** Dex: A Semantic-Graph Differencing Tool for Studying Changes in Large Code Bases // Proceedings of 20th IEEE International Conference on Software Maintenance (ICSM'04). Chicago, Illinois, September 11—14, 2004. P. 188—197.
7. **Maletic J. I., Collard M. L.** Supporting Source Code Difference Analysis // Proceedings of IEEE International Conference on Software Maintenance (ICSM'04). Chicago, Illinois. September 11—17. 2004. P. 210—219.
8. **Maitra R.** Clustering Massive Datasets With Application in Software Metrics and Tomography // Technometrics. 1 August 2001. Vol. 43. N 3. P. 336—346.
9. **Collard M. L.** Meta-Differencing: An Infrastructure for Source Code Difference Analysis. Kent State University, Kent, Ohio USA. Ph. D. Dissertation Thesis. 2004.
10. **Баргесян А. А., Куприянов М. С., Степаненко В. В., Холод И. И.** Технологии анализа данных: Data Mining, Visual Mining, Text Mining, OLAP. СПб.: БХВ-Петербург, 2007. 375 с.
11. **Мандель И. Д.** Кластерный анализ. М.: Финансы и статистика, 1988. 176 с.
12. **Фаулер М.** Рефакторинг: улучшение существующего кода. СПб.: Символ-Плюс, 2003. 432 с.
13. **Орлов С. А.** Технологии разработки программного обеспечения: Учеб. для вузов. СПб.: Питер. 2004. 528 с.
14. **Липаев В. В.** Выбор и оценивание характеристик качества программных средств: Методы и стандарты. М.: Синтег. 2001. 224 с.
15. **McCabe T. J.** A Complexity Measure // IEEE Trans SE-2, December 1976. N 4.
16. **Collins-Sussman B., Fitzpatrick B. W., Pilato M.** Version Control with Subversion. O'Relly. 2004. http://svnbook.red-bean.com/.
17. **MSqared** Resource Standard Metrics. Code Analysis for Code Reviews and Acceptance. http://msquaredtechnologies.com/m2rsm/index.htm.
18. **Navi-Manager** Vessel Monitoring System. http://www.transas.com/products/shorebased/manager/. http://www.transas.ru/products/shorebased/fleet/navi-manager/
19. **MSqared** Resource Standard Metrics Documentation. http://msquaredtechnologies.com/m2rsm/docs/rsm_metrics _narration.htm.
20. **Cohejn J.** A Coefficient of Agreement for Nominal Scales // Educational and Psychological Measurement. 1960. P. 37—46.
21. **Elemam K.** Benchmarking Kappa for Software Process Assessment Reliability Studies // Technical Report ISERN-98-02, International Software Engineering Research Network. 1998. http://citeseer.ist.psu.edu/elemam98benchmarking.html
22. **Zimmermann T., Wei.gerber P., Diehl S., Zeller A.** Mining Version Histories to Guide Software Changes // Proceedings of 26th International Conference on Software Engineering (ICSE'04). Edinburgh, Scotland, United Kingdom, May 23—28. 2004. P. 563—572.

ПРИКЛАДНЫЕ ИНФОРМАЦИОННЫЕ СИСТЕМЫ

УДК 004.896:621.771

С. М. Кулаков, д-р техн. наук, **В. Б. Трофимов**,
Н. Ф. Бондарь, канд. техн. наук, доц., **С. В. Чабан**
Сибирский государственный индустриальный университет, г. Новокузнецк

Интеллектуальная система распознавания поверхностных дефектов проката

Рассматривается актуальная прикладная задача распознавания дефектов поверхности стального проката на примере рельсов в процессе их производства. Для ее решения предложена комбинированная процедура, объединяющая аппарат искусственных нейронных сетей и динамических экспертных систем.

Введение

Рассмотрим особенности трактовки понятия "распознавание" по работам [1—6]. Согласно [1], задача распознавания образов заключается в классификации изображений (" ... образом является изображение") на основе определенных требований, причем изображения, относящиеся к одному классу образов, обладают относительно высокой степенью близости. Распознавание представляет собой классификацию на множестве признаков, оцениваемых по наблюдаемому изображению. Процесс отбора информативных при-