УДК 004.931 DOI: 10.18698/2541-8009-2019-02-442

МЕТОДИКА ПОДГОТОВКИ И РАСПОЗНАВАНИЯ ПАТТЕРНОВ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ С ИСПОЛЬЗОВАНИЕМ МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

Шакер Алаа eng.alaashaker4@gmail.com
SPIN-код: 3468-3863

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Распознавание паттернов проектирования в исходном программном коде информационной системы является актуальной задачей в области автоматического анализа и разработки программного обеспечения. В данной работе используются методы машинного обучения для автоматического обнаружения паттернов проектирования. Предлагаемая методика позволяет подготавливать, моделировать и распознавать паттерны проектирования, состоящие из произвольного количества классов. Описаны эксперименты по распознаванию пяти паттернов проектирования (синглтон, адаптер, компоновщик, декоратор, фабричный метод), выполнено сравнение результатов распознавания, полученных с помощью различных моделей машинного обучения. При этом были использованы алгоритмы K ближайших соседей, линейные модели, модели деревьев решений с градиентным усилением.

Ключевые слова

Проектирование, микроструктура, паттерн, распознавание паттернов проектирования, машинное обучение, программное обеспечение, реверс-инжиниринг, разработка программного обеспечения

Поступила в редакцию 05.02.2019

Введение. Необходимость использования паттернов в современных технологиях разработки программного обеспечения определяется факторами ускорения процессов проектирования нового программного обеспечения, эффективного сохранения удачных решений и новыми путями развития программного обеспечения, в том числе имеющего интеллектуальные модули [1, 2]. В общем случае паттерн проектирования программного обеспечения имеет четыре основных элемента [3]:

1) *имя паттерна* — идентификатор, который в общем виде характеризует задачу проектирования, основные подходы к проектированию и результаты использования паттерна;

2) *задача проектирования*, которая задает контекст проектирования и правила применения паттерна;

3) *основные подходы к проектированию*, в которых описываются структура паттерна и протоколы его взаимодействия с другими паттернами. Данные подходы не являются конкретными решениями, поскольку паттерн может использоваться в совершенно различных программных продуктах;

4) *результаты использования* — главные результаты и возможные компромиссы применения паттерна. Данный элемент необходим для понимания преимуществ и недостатков применения паттерна.

Одним из самых популярных типов паттернов проектирования программного обеспечения являются структурные паттерны, которые описывают, как классы и объекты могут объединяться для формирования более крупных структур [4]. Структурные паттерны, основанные на классах, описывают, как наследование может использоваться для создания более эффективных программных интерфейсов. Паттерны объектов описывают, как объекты могут быть скомпонованы в более крупные структуры с использованием композиции объекта или путем включения объектов в другие объекты. Порождающие паттерны проектирования определяют общие способы коммуникации между объектами и концентрируются на распределении обязанностей между ними, что дает не только лучшее понимание процессов работы программы, но и наглядное представление подходов к ее верификации.

Паттерны проектирования имеют важность также и для технологий реверс-инжиниринга, где методы автоматического распознавания структур кода являются ценным средством не только для анализа качества спроектированной системы, но и для ускорения процедур нахождения структур, которые могут использоваться повторно [5, 6]. В контексте современной обратной инженерии особый интерес представляют два направления исследований: распознавание паттернов проектирования (DPD) и реконструкция архитектуры программного обеспечения (SAR) [7, 8].

В данной работе предполагается, что уже существуют микроструктуры для исходного кода, в котором планируется провести процедуру DPD. Для непосредственного распознавания паттернов используются следующие алгоритмы и модели машинного обучения: K-ближайших соседей, линейные модели, деревья решений с градиентным усилением [9].

Микроструктуры и паттерны. Микроструктурой программного обеспечения называется любой элемент программного кода, который может быть автоматически и однозначно обнаружен из исходного кода программной системы и который представляет основные данные о структурах, составляющих систему [10]. Определение микроструктур позволяет думать о системе как о графе, где объекты кода связаны с узлами, в то время как микроструктуры связаны с ребрами. Микроструктуры являются несколько более конкретными сущностями, чем паттерны проектирования, поэтому часто используются для их формализации в конкретных приложениях.

Рассмотрим пять паттернов проектирования [11, 12], которые были использованы в данной работе и в предлагаемой методике представляются микроструктурами.

I. *Синглтон*. Является порождающим паттерном проектирования, гарантирующим, что в однопроцессном приложении будет единственный экземпляр некоторого класса, и предоставляющий глобальную точку доступа к этому экземпляру.

II. *Адаптер*. Является структурным паттерном проектирования, предназначенным для организации использования функций объекта, недоступного для модификации, через специально созданный интерфейс, то есть данный паттерн позволяет объектам с несовместимыми интерфейсами работать вместе.

III. *Компоновщик* — структурный паттерн проектирования, объединяющий объекты в древовидную структуру для представления иерархии от частного к целому. Компоновщик позволяет приложениям обращаться к отдельным объектам и к группам объектов одинаково.

IV. *Декоратор* — структурный паттерн проектирования, предназначенный для динамического подключения дополнительного поведения к объекту. Паттерн декоратор предоставляет альтернативу практике создания подклассов с целью расширения функциональности.

V. *Фабричный метод* — порождающий паттерн проектирования, предоставляющий подклассам интерфейс для создания экземпляров некоторого класса. Данный паттерн делегирует создание объектов наследникам родительского класса, что позволяет не использовать в коде программы специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

Подготовка базы данных для обучающей выборки паттернов. После выбора алгоритма машинного обучения необходимо провести его обучение. Поэтому в данной работе с учетом особенности темы исследований создана специальная база данных для обучающей выборки паттернов (рис. 1). Входными данными для алгоритма машинного обучения являются спецификации паттернов,

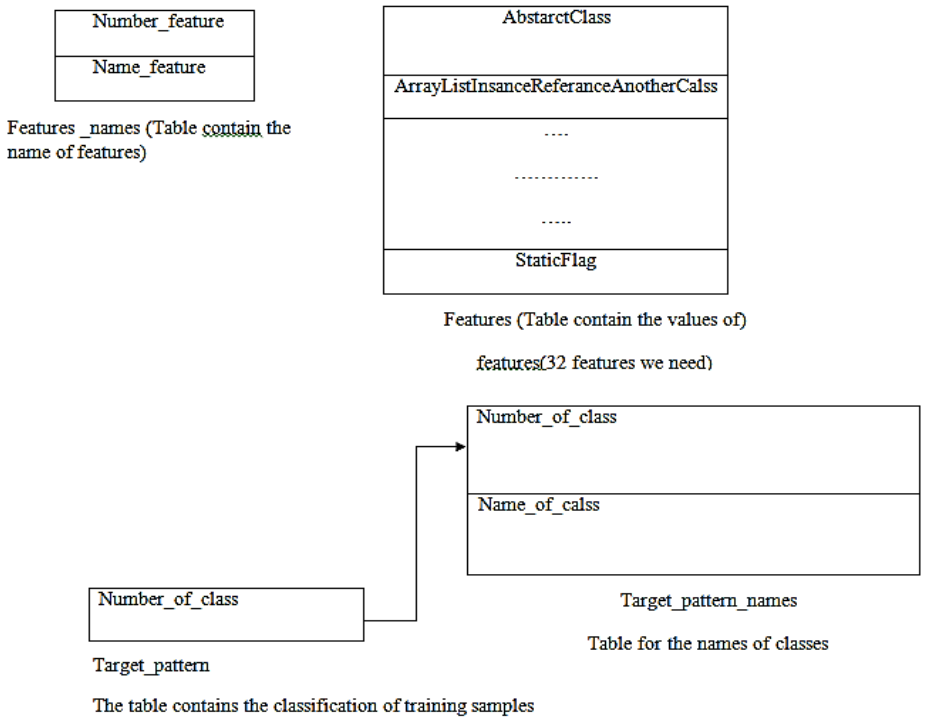

Рис. 1. Структура базы данных обучающей выборки паттернов

а выходом — определенный класс паттерна. База данных как раз включает в себя отдельную таблицу для спецификаций паттернов и таблицу для распознанных классов паттернов. В базе данных использован булевый тип данных для входных микроструктур паттернов. Первая таблица базы данных, которая называется features_names, содержит полное имя паттерна. Вторая таблица, Feautres, содержит значение спецификаций паттернов. Третья таблица, которая называется target_pattern, содержит целевой класс образца. Для получения полного имени класса создана четвертая таблица, target_pattern_name, которая содержит полные имена классов. База данных обучающих выборок заполнена 176 обучающими образцами, подготовленными вручную.

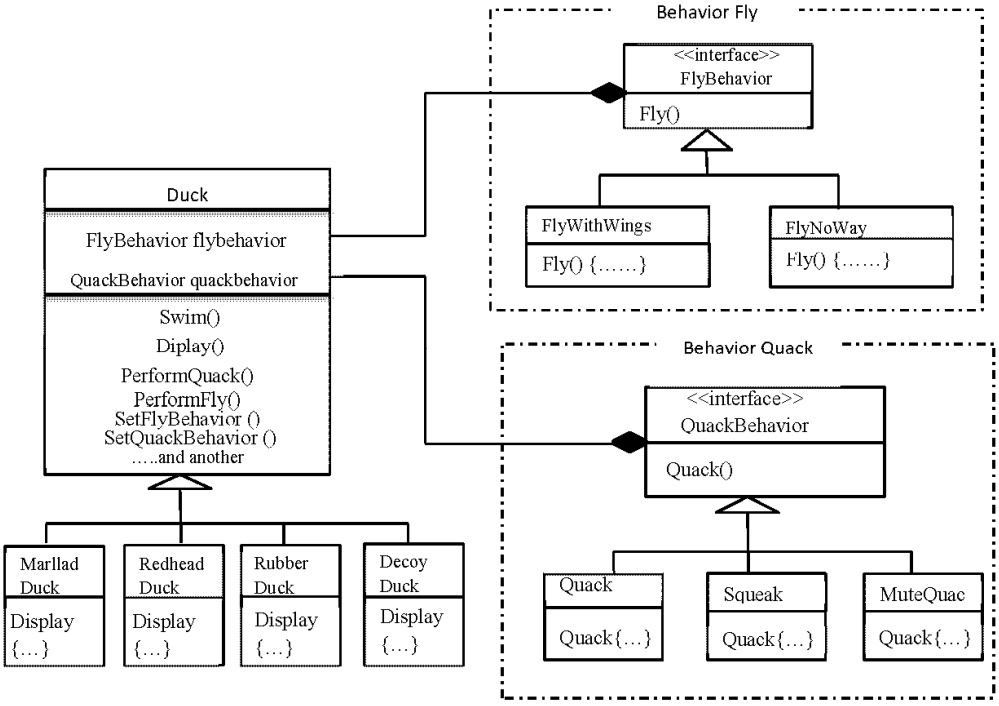

Рис. 2. Структура первого тестового образца

Эксперимент. Для проведения эксперимента были подготовлены два тестовых образца. Первый образец состоит из 10 классов, включает 3 разных паттерна и 2 типа интерфейса, первый из которых реализуется двумя классами (рис. 2). Второй тестовый образец реализован 5 классами, при этом 3 класса имеют один и тот же интерфейс (рис. 3). Первый образец имеет микроструктуру, представленную на рис. 4. Микроструктура второго образца представлена на рис. 5.

Для первого алгоритма машинного обучения K-ближайших соседей было проведено его обучение и тестирование на обучающей выборке (рис. 6).

Результат тестирования двух специально подготовленных тестовых образцов представлен на рис. 7. Два тестовых образца были успешно распознаны.

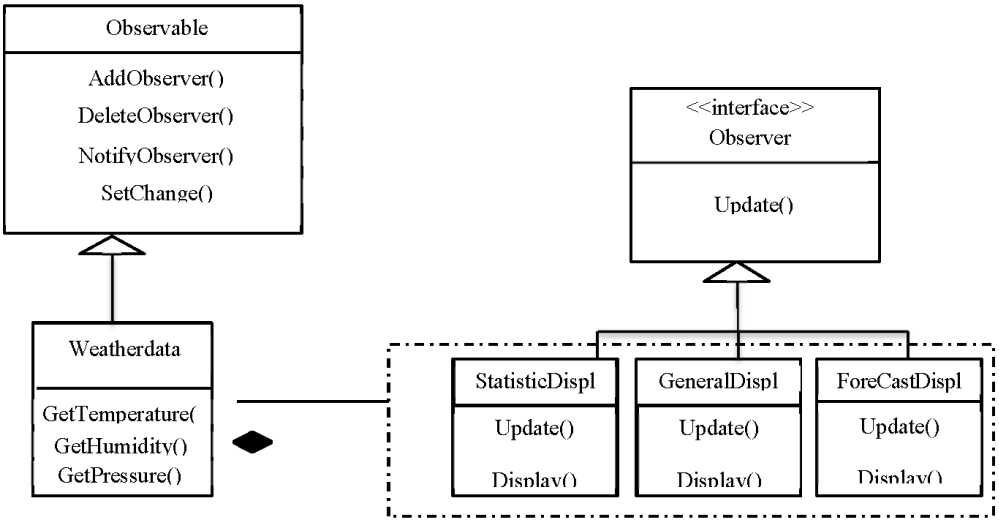

Рис. 3. Структура второго тестового образца

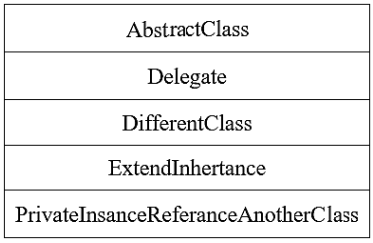

Рис. 4. Микроструктура первого тестового образца

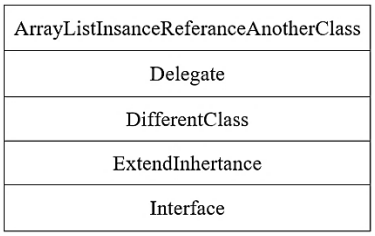

Рис. 5. Микроструктура второго тестового образца

```
Training set score(one neighbors): 1.00
Test set score (one neighbors): 1.00
Training set score(three neighbors): 0.98
Test set score (three neighbors): 1.00
```

Рис. 6. Результат обучения алгоритма K-ближайших соседей

```
result of  (one neighbors): [3 3]
result of (three neighbors): [3 3]
```

Рис. 7. Результаты распознавания алгоритмом K-ближайших соседей

Для следующего алгоритма, который представляет собой линейные модели, были проделаны те же шаги с тестированием и обучением (рис. 8). Результат работы алгоритма представлен на рис. 9. Видно, что система распознает второй паттерн, выводя выражение 3.06076882e + 00, он близок к 3 (идентификатору распознаваемого паттерна). Но первое значение, которое оно дает, составляет — 1.46389005e + 12, и оно не близко к 3, а это означает, что система не распознала первый образец.

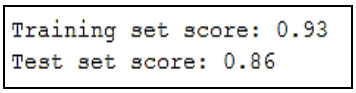

Рис. 8. Результат обучения линейными моделями

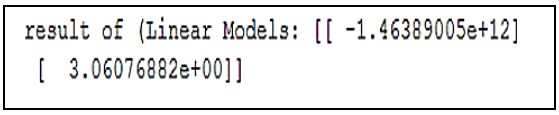

Рис. 9. Результат распознавания линейными моделями

Следующий алгоритм, который был протестирован, — деревья решений с градиентным усилением (рис. 10). Результат работы алгоритма представлен на рис. 11.

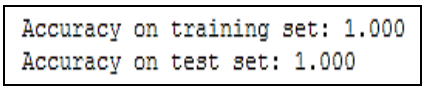

Рис. 10. Результат обучения деревьев решений с градиентным усилением

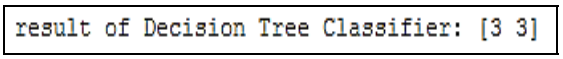

Рис. 11. Результат распознавания деревьев решений с градиентным усилением

Несмотря на успешное решение задачи, основным недостатком деревьев решений остается низкая способность к обобщению данных даже с учетом их предварительной фильтрации.

Заключение. В данной работе показано применение алгоритмов машинного обучения к проблеме распознавания паттернов проектирования, экспериментально протестировано распознавание пяти паттернов проектирования (синглтон, адаптер, компоновщик, декоратор, фабричный метод). Были использованы разные типы алгоритмов машинного обучения: K-ближайших соседей, линейные модели, деревья решений с градиентным усилением. (Отметим, что проведенный эксперимент не является полностью объективным, поскольку существует скрытая зависимость результата от вручную подготовленных обучающих образцов — паттернов проектирования).

Однако в среднем линейная модель очень плохо пригодна для данных задач. Алгоритм K-ближайших соседей справляется лучше, но при более сложных тестовых образцах также приводил к существенным ошибкам, связанным с неспособностью алгоритма автоматически разделять границы сложных классов. Для алгоритмов, основанных на деревьях решений, можно отметить, что деревья решений с градиентным усилением справляются с поставленной задачей, но имеются сложности при работе с выборкой большего размера. Поэтому в качестве продолжения работы планируется увеличить выборку, использовать автоматизированные средства синтаксического анализа кода для создания микроструктур кода, применить методы глубокого обучения для распознавания паттернов проектирования.

Литература

[1] Fowler M. Refactoring: improving the design of existing code. Addison-Wesley Professional, 2018.

[2] Morozov A.A., Polupanov A.F., Antciperov V.E., et al. Development of concurrent object-oriented logic programming system to intelligent monitoring of anomalous human activities. *Proc. 7th BIOSTEC*, 2014. Angers, France, pp. 53–62.

[3] Mall R. Fundamentals of software engineering. PHI Learning Pvt. Ltd., 2018.
[4] Fontana F.A., Caracciolo A., Zanoni M. DPB: a benchmark for design pattern detection tools. *Proc. 6th Europ. Conf. Software Maintenance and Reengineering*, 2012, pp. 235–244. DOI: 10.1109/CSMR.2012.32 URL: https://ieeexplore.ieee.org/document/6178890
[5] Dwivedi A.K., Rath S.K., Satapathy S.M., et al. Applying reverse engineering techniques to analyze design patterns in source code. *Proc. ICACCI*, 2018, pp. 1398–1404. DOI: 10.1109/ICACCI.2018.8554519 URL: https://ieeexplore.ieee.org/document/8554519
[6] Девятков В.В., Алфимцев А.Н. Нечеткая конечно-автоматная модель интеллектуального мультимодального интерфейса. *Проблемы управления*, 2011, № 2, с. 69–77.
[7] Mo R., Cai Ya., Kazman R., et al. Hotspot patterns: the formal definition and automatic detection of architecture smells. *12th Working IEEE/IFIP Conf. Software Architecture*, 2015, pp. 51–60. DOI: 10.1109/WICSA.2015.12 URL: https://ieeexplore.ieee.org/document/7158503
[8] Сакулин С.А., Алфимцев А.Н. К вопросу о практическом применении нечетких мер и интеграла Шоке. *Инженерный журнал: наука и инновации*, 2012, № 1. DOI: 10.18698/2308-6033-2012-1-71 URL: http://engjournal.ru/catalog/it/hidden/71.html
[9] Шалев-Шварц Ш., Бен-Давид Ш. Идеи машинного обучения: от теории к алгоритмам. М., ДМК Пресс, 2019.
[10] Alhusain S., Coupland S., John R., et al. Towards machine learning based design pattern recognition. *UKCI*, 2013, pp. 244–251. DOI: 10.1109/UKCI.2013.6651312 URL: https://ieeexplore.ieee.org/document/6651312
[11] Фримен Э., Сьерра К., Бейтс Б. Паттерны проектирования. СПб., Питер, 2011.
[12] Smith J.M.C., Stotts D. Elemental design patterns: a formal semantics for composition of OO software architecture. *Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop,* 2002, pp. 183–190. DOI: 10.1109/SEW.2002.1199472 URL: https://ieeexplore.ieee.org/document/1199472

Шакер Алаа — студент магистратуры кафедры «Информационные системы и телекоммуникации», МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

Научный руководитель — Алфимцев Александр Николаевич, доктор технических наук, профессор кафедры «Информационные системы и телекоммуникации» МГТУ им. Н.Э. Баумана, Москва, Российская Федерация.

METHODS OF PREPARING AND RECOGNIZING SOFTWARE PATTERNS USING MACHINE LEARNING MODELS

Alaa Shaker eng.alaashaker4@gmail.com
SPIN-code: 3468-3863

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

The paper is concerned that recognition of design patterns in the source code of the information system is an urgent task in the field of automatic analysis and software development. The authors use the machine learning methods to detect automatically design patterns. The proposed method allows you to prepare, simulate and recognize design patterns, consisting of an arbitrary number of classes. In the present paper, the authors described the experiments on the recognition of five design patterns (singleton, adapter, linker, decorator, factory method), and carried out a comparison of recognition results obtained using various machine learning models. At the same time, K nearest-neighbor algorithms, linear models, models of decision trees with gradient amplification were used.

Keywords

Design, microstructure, pattern, design pattern recognition, machine learning, software, reverse engineering, software development

Received 05.02.2019

References

[1] Fowler M. Refactoring: improving the design of existing code. Addison-Wesley Professional, 2018.

[2] Morozov A.A., Polupanov A.F., Antciperov V.E., et al. Development of concurrent object-oriented logic programming system to intelligent monitoring of anomalous human activities. *Proc. 7th BIOSTEC*, 2014. Angers, France, pp. 53–62.

[3] Mall R. Fundamentals of software engineering. PHI Learning Pvt. Ltd., 2018.

[4] Fontana F.A., Caracciolo A., Zanoni M. DPB: A benchmark for design pattern detection tools. *Proc. 6th Europ. Conf. Software Maintenance and Reengineering*, 2012, pp. 235–244. DOI: 10.1109/CSMR.2012.32 URL: https://ieeexplore.ieee.org/document/6178890

[5] Dwivedi A.K., Rath S.K., Satapathy S.M., et al. Applying reverse engineering techniques to analyze design patterns in source code. *Proc. ICACCI*, 2018, pp. 1398–1404. DOI: 10.1109/ICACCI.2018.8554519 URL: https://ieeexplore.ieee.org/document/8554519

[6] Devyatkov V.V., Alfimtsev A.N. Fuzzy finite state model of intelligent multimodal interface. *Problemy upravleniya* [Control Sciences], 2011, no. 2, pp. 69–77 (in Russ.).

[7] Mo R., Cai Ya., Kazman R., et al. Hotspot patterns: The formal definition and automatic detection of architecture smells. *12th Working IEEE/IFIP Conf. Software Architecture*, 2015, pp. 51–60. DOI: 10.1109/WICSA.2015.12 URL: https://ieeexplore.ieee.org/document/7158503

[8] Sakulin S.A., Alfimtsev A.N. To the problem on practical application of fuzzy measures and Choquet integral. *Inzhenernyy zhurnal: nauka i innovatsii* [Engineering Journal: Science and Innovation], 2012, no. 1. DOI: 10.18698/2308-6033-2012-1-71 URL: http://engjournal.ru/catalog/it/hidden/71.html (in Russ.).

[9] Shalev-Shwartz Sh., Ben-David Sh. Understanding machine learning. From theory to algorithms. Cambridge University Press, 2014. (Russ. ed.: Idei mashinnogo obucheniya: ot teorii k algoritmam. Moscow, DMK Press Publ., 2019.)
[10] Alhusain S., Coupland S., John R., et al. Towards machine learning based design pattern recognition. *UKCI*, 2013, pp. 244–251. DOI: 10.1109/UKCI.2013.6651312 URL: https://ieeexplore.ieee.org/document/6651312
[11] Freeman E., Bates B., Sierra K., et al. Head first design patterns: a brain-friendly guide. O'Reilly Media, 2004. (Russ. ed.: Patterny proektirovaniya. Sankt-Petersburg, 2011.)
[12] Smith J.M.C., Stotts D. Elemental design patterns: a formal semantics for composition of OO software architecture. *Proc. 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, 2002, pp. 183–190. DOI: 10.1109/SEW.2002.1199472 URL: https://ieeexplore.ieee.org/document/1199472

Alaa Shaker — Master's Degree Student, Department of Information Systems and Telecommunications, Bauman Moscow State Technical University, Moscow, Russian Federation.

Scientific advisor — A.N. Alfimtsev, Dr. Sc. (Eng), Professor, Department of Information Systems and Telecommunications, Bauman Moscow State Technical University, Moscow, Russian Federation.