НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
март–апрель 2026 Том 26 № 2 http://ntv.ifmo.ru/
SCIENTIFIC AND TECHNICAL JOURNAL OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS
March–April 2026 Vol. 26 No 2 http://ntv.ifmo.ru/en/
ISSN 2226-1494 (print) ISSN 2500-0373 (online)

НАУЧНО-ТЕХНИЧЕСКИЙ ВЕСТНИК
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

doi: 10.17586/2226-1494-2026-26-2-324-330

Automatic detection of software design patterns using a language model on transformer architecture

Jameleh Asaad[1]✉, Elena Yu. Avksentieva[2]

[1,2] ITMO University, Saint Petersburg, 197101, Russian Federation

[1] Jamelehasaad@gmail.com✉, https://orcid.org/0000-0003-1151-2274
[2] eavksenteva@itmo.ru, https://orcid.org/0000-0001-5000-4868

Abstract

This article addresses the significance of Gang of Four (GoF) design patterns as formal architectural solutions in object-oriented programming and emphasizes the importance of their automated detection in modern software systems. This study examines the challenges of identifying architectural solutions in extensive software systems and the constraints of conventional analytical approaches. The scientific novelty of the suggested method is in the utilization of contemporary transformer-based language models trained on source code integrated with conventional machine learning techniques for identifying structural patterns. The proposed approach employs the DeepSeek-Coder-V2 framework to generate multidimensional vector representations (embeddings) of code segments. We employ Principal Component Analysis to reduce dimensionality. The resultant embeddings serve as features for training and testing various classifiers, encompassing both linear and nonlinear models. The objective is to autonomously identify design trends. We developed a bespoke annotated dataset of 23 GoF patterns and additional architectural patterns derived from actual open-source projects. Experiments demonstrate that transformer-based code embeddings significantly outperform conventional feature extraction techniques, achieving a macro-averaged F1-score of up to 0.82. The test demonstrates that the embeddings accurately represent both the syntactic and semantic characteristics of the source code. The proposed strategy is more versatile and capable of managing a broader array of scenarios compared to manual or heuristic-based solutions. It functions effectively for pattern recognition tasks and can be utilized to analyze extensive codebases. Potential applications encompass rectifying, sustaining, and enhancing the quality and comprehension of software architecture. This approach establishes a unified framework for subsequent research and advancement in software engineering.

Keywords

code embeddings, DeepSeek, design pattern detection, gang of four, machine learning, Java, feature extraction, software engineering

For citation: Asaad J., Avksentieva E.Yu. Automatic detection of software design patterns using a language model on transformer architecture. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2026, vol. 26, no. 2, pp. 324–330. doi: 10.17586/2226-1494-2026-26-2-324-330

УДК 004.852

Автоматическое обнаружение паттернов проектирования программного обеспечения с использованием языковой модели, основанной на архитектуре трансформера

Жамилех Асаад[1]✉, Елена Юрьевна Авксентьева[2]

[1,2] Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация

[1] Jamelehasaad@gmail.com✉, https://orcid.org/0000-0003-1151-2274
[2] eavksenteva@itmo.ru, https://orcid.org/0000-0001-5000-4868

Аннотация

Введение. Определена значимость паттернов проектирования Gang of Four (GoF), являющихся формализованными архитектурными решениями в объектно-ориентированном программировании. Обоснована актуальность автоматизированного обнаружения данных паттернов в исходном коде современных программных продуктов. Рассмотрены проблемы выявления архитектурных шаблонов в программном обеспечении и

ограничения традиционных аналитических подходов. Научная новизна предлагаемого метода заключается в использовании современных языковых моделей на основе трансформеров, обученных на исходном программном коде в сочетании с традиционными методами машинного обучения для выявления структурных паттернов. **Метод.** Предложено использовать фреймворк DeepSeek-Coder-V2 для генерации многомерных векторных представлений (эмбеддингов) сегментов кода, и для уменьшения их размерности применить анализ главных компонент. Полученные векторные представления используются в качестве признаков для обучения и тестирования классификаторов, включающих линейные и нелинейные модели с целью автоматического распознавания паттернов проектирования. **Основные результаты.** Создан и размечен набор данных, включающий 23 шаблона GoF и несколько дополнительных архитектурных шаблонов, собранный из реальных проектов с открытым исходным кодом. Эксперименты показали, что трансформерные представления кода значительно превосходят альтернативные методы извлечения признаков, достигая макро-усредненного значения F1-меры 0,82. Проведенная оценка подтверждает, что эмбеддинги успешно отражают синтаксические и семантические особенности кода. **Обсуждение.** В отличие от используемых аналитических и эвристических методов, предложенный метод более масштабируем и адаптируем к различным контекстам. Эксперименты продемонстрировали его высокую эффективность на задаче распознавания шаблонов и применимость для анализа больших кодовых баз с целью рефакторинга и сопровождения программного обеспечения.

Ключевые слова

векторные представления кода, DeepSeek, обнаружение шаблонов проектирования, банда четырех, машинное обучение, Java, извлечение признаков, программная инженерия

Ссылка для цитирования: Асаад Ж., Авксентьева Е.Ю. Автоматическое обнаружение паттернов проектирования программного обеспечения с использованием языковой модели, основанной на архитектуре трансформера // Научно-технический вестник информационных технологий, механики и оптики. 2026. Т. 26, № 2. С. 324–330 (на англ. яз.). doi: 10.17586/2226-1494-2026-26-2-324-330

Introduction

Design pattern detection has become one of the most notable and practical applications of machine learning in software engineering [1]. Since at least 2001 [2], researchers have shown growing interest in automatically identifying design patterns in source code, and this interest continues today. This ongoing focus underscores design pattern recognition as a dynamic research domain, motivated by both theoretical significance and practical necessity. Nonetheless, continuous contributions to this domain are driven by factors beyond academic trends. Identifying design patterns is crucial for comprehending and sustaining software systems. It assists developers in recognizing reusable solutions within code, which can function as structural maps or guidance during activities like as extension, refactoring, or debugging, particularly in inadequately documented or outdated systems [3]. By identifying design patterns, developers may uphold compliance with object-oriented design principles and prevent the introduction of structural defects [4]. Furthermore, feature extraction, which underpins design pattern recognition, is an essential element in a wider array of code analysis applications. It facilitates activities including bug discovery, vulnerability research, and automated code production [5]. Consequently, augmenting the quality and efficacy of feature extraction methods directly improves several aspects of intelligent software analysis [6]. Manual feature extraction necessitates extensive domain expertise and a profound comprehension of object-oriented design principles. The method is both time-intensive and requires considerable effort and financial investment [7]. Moreover, it is frequently susceptible to errors and challenging to scale, particularly in extensive or inadequately documented codebases [8]. The incorporation of machine learning into software engineering, especially in feature extraction, produces significant advantages [9]. Machine learning techniques automate feature extraction, so reducing manual intervention, lowering costs, and improving the efficiency and consistency of design pattern discovery across large codebases.

This automation not only expedites the analysis but also improves the reliability of pattern recognition in complex software systems. Design patterns in software engineering, commonly referred to as Gang of Four (GoF) patterns, are standardized [10] solutions to recurring problems in software design. They are crucial for facilitating communication among developers by providing a shared language and framework [8]. “GoF” denotes “Gang of Four” referring to the four authors — Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides — who authored the seminal book “Design Patterns: Elements of Reusable Object-Oriented Software” in 1994 [11]. The original GoF library has 23 design patterns categorized into three primary groups: behavioral, structural, and creational. Despite the emergence of new design patterns throughout time, the GoF patterns remain very significant and continue to be extensively utilized and examined in contemporary software development.

Research Subject

In the context of automated design pattern detection, feature extraction plays a central role. It refers to the process of transforming raw data into a collection of quantifiable attributes, or “features” that may serve as input for machine learning models [12]. In essence, it involves identifying and isolating the most significant elements within a dataset that assists a model in discerning patterns, making decisions, or performing classification or predictive tasks [7]. This stage is crucial as most algorithms cannot directly utilize raw data (such as text, photos, or source code) due to its complexity or lack of organization [13]. In image processing, features may include edges, textures, or color histograms [14]. In text analysis, characteristics may include word frequencies, *n*-grams, or embeddings.

In source code analysis, characteristics may encompass the quantity of classes, the sequence of method invocations, or the structural links among objects [15]. Effective feature extraction enhances model performance and reduces operational costs by focusing just on the most significant aspects of the data.

It is crucial for machine learning models to process and understand code because code is a rich, structured form of data that encapsulates logic, design intent, and behavior. In software engineering tasks — such as design pattern detection, code summarization, bug prediction, or automated code generation — understanding code semantics and structure enables models to learn meaningful patterns and make intelligent decisions [16]. If processing and feature extraction are inadequately executed, the model may fail to identify the genuine relationships among program components, resulting in suboptimal performance and unreliable predictions. Machine learning systems may bridge the divide between raw syntax and advanced design knowledge by extracting valuable aspects from code, either manually or via learnt representations. This enhances their accuracy, scalability, and applicability for practical software analysis tasks. The evolution of machine learning techniques for code analysis, particularly in relation to design patterns, has been significant [17]. Over the past decade, the application of machine learning for code analysis has undergone significant transformation. This is due to the increasing prevalence of large-scale code repositories and the emergence of novel methods for program representation. Machine learning models capable of discerning intricate patterns directly from source code are gradually supplanting or augmenting conventional rule-based or manually crafted approaches, which frequently required domain expertise and were susceptible to syntactical alterations. This alteration is particularly significant in the context of identifying design patterns [16]. Initial models predominantly relied on static analysis tools and established heuristics, but contemporary models include graph-based representations, Abstract Syntax Trees (ASTs), and Code Embeddings to more effectively encapsulate structural and semantic information. Deep learning, particularly transformer-based models trained on source code, has enabled more precise identification of design patterns applicable to extensive codebases, even in instances with inadequate documentation or diverse architectures. This modification facilitates the automation of software analysis tasks and introduces new opportunities for intelligent development tools that assist with maintenance, refactoring, and comprehension. Recent advancements in identifying design patterns have increasingly depended on automated feature extraction and machine learning to address issues of scalability, accuracy, and generalizability. One of the early contributions in this domain is the work of Thaller et al. [9] who introduced the Feature Maps approach. This method encodes software systems as normalized feature-role maps that link design pattern roles to software entities using interpretable numerical features. The generated maps serve as inputs to classifiers, such as random forests or convolutional neural networks, enabling the detection of design patterns in a manner that remains comprehensible to developers. Zanoni et al. [15] proposed Metrics and Architecture Reconstruction PLug-in for Eclipse and Design Pattern Detection (MARPLE-DPD) which integrates structural features derived from static analysis with machine learning classifiers. Their method enhances generalization across systems by enabling the detection of variable-class and incomplete pattern instances, showing practical viability on real-world software projects. In parallel, Nazar et al. [18] presented Feature-Based Design Pattern Detection (DPDF), a feature-based method that utilizes Word2Vec to embed semantic information from Java code tokens. These embeddings are used to construct geometric representations of classes which are then classified to identify design patterns. DPDF demonstrated high precision and recall, especially for behavioral and creational patterns. Collectively, these studies illustrate a shift towards using more sophisticated representations and learning paradigms, with a strong emphasis on feature engineering and model interpretability in advancing automated design pattern detection.

This study presents a dual scientific innovation. Initially, we develop and publicly disseminate a comprehensive, annotated dataset “GoF-DS” that encompasses all 23 original GoF design patterns, in addition to supplementary non-GoF architectural patterns, sourced from actual open-source Java projects. This dataset facilitates empirical assessment on a more extensive and representative array of patterns than most previous studies. Secondly, we present a hybrid approach that amalgamates DeepSeek-Coder-V2, an advanced transformer-based language model trained on code, with traditional machine learning classifiers. Dense semantic embeddings are produced from raw Java source files and enhanced via dimensionality reduction methods like Principal Component Analysis (PCA). These embeddings are subsequently employed to train a range of classifiers, facilitating automated, scalable, and precise pattern recognition.

Our method presents multiple advantages over current techniques: it obviates manual feature engineering, generalizes across varied codebases, and proficiently captures both syntactic and semantic attributes of source code. This methodology offers a cohesive and adaptable framework for practical applications in software maintenance, refactoring, reverse engineering, and architectural evaluation.

Methodology

This study proposes a method that integrates transformer-based code embeddings with classical machine learning algorithms for software design pattern detection. We utilized DeepSeek-Coder-V2, a large-scale language model pretrained on source code, to extract semantic vector representations from Java source files. Code snippets were passed directly into the model using multiple embedding dimensions — 64, 128, 256, 300, 350, and 512 — to examine the effect of embedding size on model performance. No preprocessing or tokenization was performed; rather, the unaltered source code was utilized to preserve its original structure and grammar. Subsequently, we employed the embeddings to train several

machine learning classifiers, including Support Vector Machines (SVC and Linear SVC), Gaussian Naive Bayes, Logistic Regression, Ridge Classifier, Stochastic Gradient Descent (SGD), and a Multi-Layer Perceptron (MLP) for a maximum of 1,000 iterations. We employed macro-averaged accuracy, precision, and F1-score to evaluate the performance of the models. This ensured equitable comparison across various design pattern categories.

The GoF design pattern dataset was compiled from GitHub and contains annotated Java code samples for 23 distinct GoF patterns. We utilized the GitHub API to develop an automated workflow that incorporates additional non-GoF patterns into this dataset. This pipeline contains Java code that aligns with architectural patterns, such as Event Sourcing, Command Query Responsibility Segregation, Microservices, Domain-Driven Design, and Service Locator. We used the phrase 'design pattern' in combination with pattern-specific terms to perform the searches, yielding results exclusively including Java code. We conducted numerous scans of repositories to obtain Java assets. Subsequently, we assessed their pattern significance by keyword comparison and AST analysis. Validated files were then organized into directories according to their patterns. A limit of 150 files per pattern was established to maintain class balance and prevent redundancy. Repositories that have already undergone scanning were prohibited from being rescanned. This method yielded a curated and validated dataset of both GoF and non-GoF patterns, enabling a comprehensive comparison of detection approaches. The resulting dataset, named GoF-DS, comprises 3,418 Java source files totaling approximately 81,296 Lines Of Code. Using the deepseek-coder-v2:latest model (8.9 GB), six distinct embedding sets were generated with dimensions of 64, 128, 256, 300, 350, and 512.

Embedding-Based Feature Extraction and Dimensionality Optimization: This research employed DeepSeek-Coder-V2, a transformer model designed for code comprehension, to extract features. It processes unrefined Java source code, encompassing comments and formatting, without any preparation or tokenization, hence preserving the syntactic and semantic structure. The approach generates dense vector embeddings that encapsulate both semantic significance and structural characteristics. This facilitates the identification of logical processes and design pattern attributes more effectively than conventional handmade or token-based features. We examined embedding dimensions of 64, 128, 256, 300, 350, and 512 to evaluate their impact on the equilibrium between representational capability and computational expense. The 512-dimensional embeddings had the greatest Mutual Information scores, reaching around 0.141 and maintaining elevated levels across features. This indicates they possess superior discernment for categorization tasks. Conversely, lower-dimensional embeddings declined more significantly. By the 20th feature, the 64-dimensional dataset had decreased to 0.074. PCA was employed on the embeddings to enhance computational efficiency while preserving significant variation. Ultimately, 425 principal components were selected for further tests since they preserved 99 % of the data variation while reducing complexity and minimizing the risk of overfitting.

The extracted embeddings were used to train a broad selection of classical machine learning models to perform multi-class classification of design patterns. This included both linear and non-linear classifiers. Among the neural approaches, an MLP was employed to model non-linear relationships in the data. We considered linear models such as Logistic Regression (including its class-weight-balanced variant) and the Ridge Classifier due to its user-friendliness, speed, and capacity for data regularization. We also examined discriminative modeling with Linear Discriminant Analysis (LDA) and several SVM techniques. These encompassed Linear SVC, SGD, conventional SVM, its balanced variant, and Nu-Support Vector Classification (NuSVC), which provides an alternative method of regularization. These are effective for handling high-dimensional and sparse datasets. We employed a Random Forest classifier as a robust ensemble model because of its capacity to manage non-linear interactions and reduce variation through averaging. We employed the scikit-learn toolkit to construct all models, training each individually on each embedding dimension to ensure uniform conditions and reproducibility of outcomes. We employed stratified cross-validation to ensure that the assessment was equitable and dependable. This maintained consistent class distributions across all fields. The evaluation parameters employed encompassed accuracy, F1-score, and macro-averaged precision. We selected these measures to compensate for the impact of class imbalance, as macro-averaging assigns equal weight to each class, regardless of its frequency. This strategy ensured equitable evaluation of all 24 classes.

Results and Discussion

The assessment of diverse machine learning models, as illustrated in Table 1, uncovered significant insights on the attributes of the gathered dataset. The MLP and Neural Network models exhibited superior performance across all criteria, achieving an accuracy of 0.79 and an F1-score of 0.78, respectively. This indicates the presence of intricate, non-linear correlations within the data that deep learning methodologies can adeptly discern.

Linear models, such as Logistic Regression, LDA (both default and shrinkage-based variants), and Linear SVC, also demonstrated strong and consistent results (accuracy range 0.74–0.77), suggesting that the dataset exhibits partially linear separability. Furthermore, the relatively poor performance of the Random Forest classifier (accuracy 0.49) and the improvement observed when using balanced variants of certain models (e.g., Logistic Regression Balanced and SVM Balanced) imply a class imbalance within the dataset. Overall, the results indicate that the dataset is of high quality, containing meaningful and learnable patterns, yet also presenting a degree of imbalance that benefits from algorithmic adjustments.

Table 2 presents the F1-score performance of four different design pattern detection methods — Deepseek (DS-Approach), Feature Map, DPDF, and MARPLE-DPD — evaluated across a variety of GoF and non-GoF design patterns.

Table 1. Accuracy, precision, recall, and F1-score of various models

Machine Learning Model	Accuracy	Precision	Recall	F1-score
MLP	0.79	0.79	0.79	0.79
Neural Network	0.78	0.78	0.78	0.78
LDA (default, solver='svd')	0.76	0.77	0.75	0.75
Logistic Regression	0.76	0.75	0.75	0.75
Logistic Regression Balanced	0.76	0.76	0.75	0.75
Linear SVC	0.77	0.77	0.77	0.77
Ridge Classifier	0.75	0.75	0.74	0.74
SGD Classifier	0.75	0.75	0.74	0.74
LDA (lsqr, shrinkage=0.2)	0.74	0.74	0.73	0.73
NuSVC	0.65	0.67	0.64	0.65
SVM	0.63	0.64	0.62	0.62
SVM Balanced	0.64	0.65	0.63	0.63
Random Forest	0.49	0.51	0.48	0.48

The findings indicate that DS-Approach1, grounded on the Deepseek architecture, consistently excels across most patterns, with Observer (0.85), Adapter (0.83), and Visitor (0.83) achieving the greatest F1-scores. The DPDF technique demonstrates superior performance in several instances, particularly for Abstract Factory (0.93), Visitor (0.94), and Memento (0.87). This indicates a strong proficiency in identifying certain patterns in the construction of objects and human behavior. MARPLE-DPD performs effectively in specific scenarios, such as

Table 2. Performance evaluation of design pattern detection methods

Design Pattern	DS-Approach1	DS-Approach2	Feature Map [9]	DPDF [18]	MARPLE-DPD [15]
Singleton	0.79	0.86	0.66	0.74	0.72
Observer	0.85	0.85	0.49	0.85	0.51
Builder	0.67	0.79	0.61	0.83	0.55
Abstract Factory	0.83	0.84	0.52	0.93	0.76
Factory Method	0.72	0.78	0.55	0.78	0.81
Adapter	0.83	0.86	0.33	0.69	0.82
Decorator	0.77	0.78	0.23	0.78	0.59
Visitor	0.83	0.84	0.65	0.94	0.63
Bridge	0.82	0.87	—	—	—
Chain of Responsibility	0.83	0.71	—	—	—
Command	0.80	0.83	—	—	—
Composite	0.71	0.84	—	—	—
Facade	0.77	0.79	—	0.71	—
Flyweight	0.61	0.81	—	—	—
Interpreter	0.93	0.81	—	—	—
Iterator	0.88	0.91	—	—	—
Mediator	0.72	0.76	—	—	—
Memento	0.68	0.74	—	0.87	—
Prototype	0.74	0.83	—	0.82	—
Proxy	0.70	0.74	—	0.62	—
State	0.82	0.77	—	—	—
Strategy	0.89	0.77	—	—	—
Template Method	0.87	0.89	—	—	—
Non-GoF	0.91	0.95	—	—	—

Factory Method (0.81) and Adapter (0.82), but is less effective overall. Conversely, the Feature Map methodology typically yields inferior outcomes, particularly for structural patterns such as Adapter (0.33) and Decorator (0.23). This indicates difficulty in effectively capturing structural aspects. Results for DS-Approach1 are available just for certain patterns, including Bridge, Command, and Strategy. This method is also quite effective in these instances.

The findings indicate that the Deepseek-based methodology and DPDF are effective across various design patterns. The detection approach significantly influences the efficacy of classification performance. This enhancement indicates that incorporating explicit category information into the feature set aids the model in distinguishing various design patterns more effectively. Incorporating the pattern category as an additional input enhances the MLP classifier by providing semantic information that likely aligns with the structure acquired by the DeepSeek embeddings. The steady improvement of all assessment metrics — F1-score, accuracy, and recall — demonstrates that the model enhances its ability to detect both positive and negative instances effectively. The results indicate that design pattern categorization serves as a beneficial inductive bias in the learning process, particularly within transformer-based feature extraction frameworks. A column for the pattern category was introduced to evaluate if DeepSeek accurately identifies design pattern categories. This approach employs an MLP classifier referred to as DS-Approach2. We observed an enhancement of around 3 % across all performance indicators. The F1-score, accuracy, and recall all attained 82 %.

While a 3 % improvement may appear modest, it is consistent across all key evaluation metrics — F1-score, accuracy, and recall — indicating a meaningful gain in model reliability. In classification tasks involving fine-grained distinctions such as design pattern detection, even marginal improvements can reflect a better generalization of the model to unseen data. This result supports the hypothesis that integrating design pattern category information provides a helpful structural bias, enhancing the discriminative capacity of the classifier.

Conclusion

This study presents a comprehensive approach to the automated detection of GoF design patterns in Java source code through the integration of transformer-based embeddings and classical machine learning models. By leveraging DeepSeek-Coder-V2 to generate rich, high-dimensional representations of raw source code, the proposed method captures both syntactic and semantic features, without relying on manual preprocessing or handcrafted features. The introduction of the GoF-DS dataset, encompassing a diverse and balanced collection of design pattern instances, further strengthens the empirical foundation of the study. Experimental results demonstrate that the proposed embedding-based method achieves superior classification performance compared to traditional feature-based approaches, with the Multi-Layer Perceptron and Neural Network models attaining F1-scores of 0.79 and 0.78, respectively. These findings underscore the suitability of transformer-derived embeddings for complex software analysis tasks and highlight the potential of integrating modern representation learning techniques into software engineering pipelines. Moreover, the investigation into embedding dimensionality and the application of dimensionality reduction techniques, such as Principal Component Analysis, offer insights into the trade-offs between computational efficiency and model accuracy. The performance of both linear and non-linear classifiers validates the quality and learnability of the dataset, while observed class imbalances suggest the need for additional data curation or algorithmic adjustments in future work. In summary, this work contributes a validated dataset, a scalable detection pipeline, and empirical evidence supporting the use of transformer-based embeddings for design pattern classification. These contributions establish a foundation for future advancements in intelligent software maintenance, documentation, and reverse engineering, reinforcing the role of machine learning in modern software engineering practices.

Data and Code Availability

The GoF-DS dataset and all scripts for embedding extraction, model training, and evaluation are publicly available[1] for reproducibility.

[1] Available at: https://github.com/Jameleh/GoF-DS-Transformer-ML (accessed: 14.02.2026).

References

1. Albin-Amiot H., Guéhéneuc Y.-G. Meta-modeling design patterns: application to pattern detection and code synthesis. *Proc. of the 1st ECOOP Workshop on Automating Object-Oriented Software Development Methods*, 2001, pp. 1–8.
2. Albin-Amiot H., Cointe P., Gueheneuc Y.-G., Jussien N. Instantiating and detecting design patterns: Putting bits and pieces together. *Proc. of the 16th Annual International Conference on Automated Software Engineering (ASE 2001)*, 2001, pp. 166–173. https://doi.org/10.1109/ase.2001.989802
3. Kouli M., Rasoolzadegan A. A feature-based method for detecting design patterns in source code. *Symmetry*, 2022, vol. 14, no. 7, pp. 1491. https://doi.org/10.3390/sym14071491
4. Asaad J., Avksentieva E. A review of approaches to detecting software design patterns. *Proc. of the 35th Conference of Open Innovations*

Литература

1. Albin-Amiot H., Guéhéneuc Y.-G. Meta-modeling design patterns: application to pattern detection and code synthesis // Proc. of the 1st ECOOP Workshop on Automating Object-Oriented Software Development Methods. 2001. P. 1–8.
2. Albin-Amiot H., Cointe P., Gueheneuc Y.-G., Jussien N. Instantiating and detecting design patterns: Putting bits and pieces together // Proc. of the 16th Annual International Conference on Automated Software Engineering (ASE 2001). 2001. P. 166–173. https://doi.org/10.1109/ase.2001.989802
3. Kouli M., Rasoolzadegan A. A feature-based method for detecting design patterns in source code // Symmetry. 2022. V. 14. N 7. P. 1491. https://doi.org/10.3390/sym14071491
4. Asaad J., Avksentieva E. A review of approaches to detecting software design patterns // Proc. of the 35th Conference of Open Innovations

Association (FRUCT), 2024, pp. 142–148. https://doi.org/10.23919/fruct61870.2024.10516345
5. Markson R., Samson M., Owen A. Automated code review: leveraging generative AI for vulnerability detection in software development. *Author content*, 2023, pp. 1–34.
6. Mostafa S., Cynthia S.T., Roy B., Mondal D. Feature transformation for improved software bug detection and commit classification. *Journal of Systems and Software*, 2025, vol. 219, pp. 112205. https://doi.org/10.1016/j.jss.2024.112205
7. Ding S., Zhu H., Jia W., Su C. A survey on feature extraction for pattern recognition. *Artificial Intelligence Review*, 2012, vol. 37, no. 3, pp. 169–180. https://doi.org/10.1007/s10462-011-9225-y
8. Pandey S.K., Chand S., Horkoff J., Staron M., Ochodek M., Durisic D. Design pattern recognition: a study of large language models. *Empirical Software Engineering*, 2025, vol. 30, no. 3, pp. 69. https://doi.org/10.1007/s10664-025-10625-1
9. Thaller H., Linsbauer L., Egyed A. Feature maps: a comprehensible software representation for design pattern detection. *Proc. of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 207–217. https://doi.org/10.1109/saner.2019.8667978
10. Almadi S.H.S., Hooshyar D., Ahmad R.B. Bad smells of gang of four design patterns: a decade systematic literature review. *Sustainability*, 2021, vol. 13, no. 18, pp. 10256. https://doi.org/10.3390/su131810256
11. Gamma E., Helm R., Johnson R., Vlissides J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994, 416 p.
12. Casey B., Santos J.C.S., Perry G. A survey of source code representations for machine learning-based cybersecurity tasks. *ACM Computing Surveys*, 2025, vol. 57, no. 8, pp. 1–41. https://doi.org/10.1145/3721977
13. Jin D., He C., Zou Q., Qin Y., Wang B. Source code vulnerability detection based on joint graph and multimodal feature fusion. *Electronics*, 2025, vol. 14, no. 5, pp. 975. https://doi.org/10.3390/electronics14050975
14. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition. *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. https://doi.org/10.1109/cvpr.2016.90
15. Zanoni M., Fontana F.A., Stella F. On applying machine learning techniques for design pattern detection. *Journal of Systems and Software*, 2015, vol. 103, pp. 102–117. https://doi.org/10.1016/j.jss.2015.01.037
16. Alon U., Zilberstein M., Levy O., Yahav E. Code2vec: Learning distributed representations of code. *Proc. of the ACM on Programming Languages*, 2019, vol. 3, no. POPL, pp. 1–29. https://doi.org/10.1145/3290353
17. Allamanis M., Barr E.T., Devanbu P., Sutton C. A survey of machine learning for big code and naturalness. *ACM Computing Surveys*, 2019, vol. 51, no. 4, pp. 1–37. https://doi.org/10.1145/3212695
18. Nazar N., Aleti A., Zheng Y. Feature-based software design pattern detection. *Journal of Systems and Software*, 2022, vol. 185, pp. 111179. https://doi.org/10.1016/j.jss.2021.111179

Association (FRUCT). 2024. P. 142–148. https://doi.org/10.23919/fruct61870.2024.10516345
5. Markson R., Samson M., Owen A. Automated code review: leveraging generative AI for vulnerability detection in software development // Author content. 2023. P. 1–34.
6. Mostafa S., Cynthia S.T., Roy B., Mondal D. Feature transformation for improved software bug detection and commit classification // Journal of Systems and Software. 2025. V. 219. P. 112205. https://doi.org/10.1016/j.jss.2024.112205
7. Ding S., Zhu H., Jia W., Su C. A survey on feature extraction for pattern recognition // Artificial Intelligence Review. 2012. V. 37. N 3. P. 169–180. https://doi.org/10.1007/s10462-011-9225-y
8. Pandey S.K., Chand S., Horkoff J., Staron M., Ochodek M., Durisic D. Design pattern recognition: a study of large language models // Empirical Software Engineering. 2025. V. 30. N 3. P. 69. https://doi.org/10.1007/s10664-025-10625-1
9. Thaller H., Linsbauer L., Egyed A. Feature maps: a comprehensible software representation for design pattern detection // Proc. of the IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2019. P. 207–217. https://doi.org/10.1109/saner.2019.8667978
10. Almadi S.H.S., Hooshyar D., Ahmad R.B. Bad smells of gang of four design patterns: a decade systematic literature review // Sustainability. 2021. V. 13. N 18. P. 10256. https://doi.org/10.3390/su131810256
11. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994. 416 p.
12. Casey B., Santos J.C.S., Perry G. A survey of source code representations for machine learning-based cybersecurity tasks // ACM Computing Surveys. 2025. V. 57. N 8. P. 1–41. https://doi.org/10.1145/3721977
13. Jin D., He C., Zou Q., Qin Y., Wang B. Source code vulnerability detection based on joint graph and multimodal feature fusion // Electronics. 2025. V. 14. N 5. P. 975. https://doi.org/10.3390/electronics14050975
14. He K., Zhang X., Ren S., Sun J. Deep residual learning for image recognition // Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. P. 770–778. https://doi.org/10.1109/cvpr.2016.90
15. Zanoni M., Fontana F.A., Stella F. On applying machine learning techniques for design pattern detection // Journal of Systems and Software. 2015. V. 103. P. 102–117. https://doi.org/10.1016/j.jss.2015.01.037
16. Alon U., Zilberstein M., Levy O., Yahav E. Code2vec: Learning distributed representations of code // Proc. of the ACM on Programming Languages. 2019. V. 3. N POPL. P. 1–29. https://doi.org/10.1145/3290353
17. Allamanis M., Barr E.T., Devanbu P., Sutton C. A survey of machine learning for big code and naturalness // ACM Computing Surveys. 2019. V. 51. N 4. P. 1–37. https://doi.org/10.1145/3212695
18. Nazar N., Aleti A., Zheng Y. Feature-based software design pattern detection // Journal of Systems and Software. 2022. V. 185. P. 111179. https://doi.org/10.1016/j.jss.2021.111179

Authors

Jameleh Asaad — PhD Student, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 58744629900, https://orcid.org/0000-0003-1151-2274, Jamelehasaad@gmail.com
Elena Yu. Avksentieva — PhD (Education), Associate Professor, Associate Professor, ITMO University, Saint Petersburg, 197101, Russian Federation, SC 57190830859, https://orcid.org/0000-0001-5000-4868, eavksenteva@itmo.ru

Авторы

Асаад Жамилех — аспирант, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 58744629900, https://orcid.org/0000-0003-1151-2274, Jamelehasaad@gmail.com
Авксентьева Елена Юрьевна — кандидат педагогических наук, доцент, доцент, Университет ИТМО, Санкт-Петербург, 197101, Российская Федерация, SC 57190830859, https://orcid.org/0000-0001-5000-4868, eavksenteva@itmo.ru

Received 25.05.2025
Approved after reviewing 14.01.2026
Accepted 15.03.2026

Статья поступила в редакцию 25.05.2025
Одобрена после рецензирования 14.01.2026
Принята к печати 15.03.2026