Инженерный вестник Дона, №8 (2025)
ivdon.ru/ru/magazine/archive/n8y2025/10316

Разработка программного модуля для автоматической генерации кода на основе UML-диаграмм

Р.Д. Ефремов, А.А. Буткина, С.А. Фирсова, С.Д. Шибайкин, Р.А. Жарков

Мордовский государственный университет им. Н.П. Огарева, Саранск

Аннотация: В статье рассматривается разработанный авторами программный модуль, предназначенный для автоматической генерации программного кода на основе UML-диаграмм. Актуальность разработки данного модуля обусловлена ограничениями существующих зарубежных инструментов кодогенерации, связанными с функциональностью, удобством использования, поддержкой современных технологий, а также с их недоступностью на территории России. Модуль анализирует JSON-файлы, полученные экспортированием из онлайн-сервиса draw.io UML-диаграмм и преобразует их в код на выбранном языке программирования (Python, C++, Java) или в DDL-скрипты для СУБД (PostgreSQL, Oracle, MySQL). В качестве основных инструментов разработки были использованы язык Python и шаблонизатор Jinja2. Работа программного модуля продемонстрирована на примере небольшого проекта «Система управления библиотекой». В ходе исследования была проведена серия тестов по автоматической генерации кодов на основе типовых архитектур программно-информационных систем. Результаты тестов показали, что сгенерированный с помощью разработанного модуля код полностью соответствует исходным UML-диаграммам, включая структуру классов, связи между ними, а также конфигурацию базы данных и инфраструктуры (Docker Compose). Практическая значимость исследования заключается в том, что предложенная концепция генерации программного кода на основе визуальных моделей UML-диаграмм, построенных в популярном онлайн-редакторе draw.io, значительно облегчает разработку программно-информационных систем, и может быть использована в учебных целях.
Ключевые слова: генерация кода, автоматизация, python, jinja2, uml-диаграмма, json, шаблонизатор, парсинг, диаграмма классов, база данных, диаграмма развертывания.

Введение

В современных условиях стремительного технического прогресса и возрастания сложности программных продуктов высокоэффективное проектирование программного обеспечения становится основополагающим фактором успешной разработки. Унифицированный язык моделирования (UML) прочно вошел в практику разработки программного обеспечения как мощный инструмент для визуализации, проектирования и документирования систем. UML представляет собой стандартизированный набор графических элементов, каждый из которых имеет строго определённое значение, что делает UML универсальным языком, понятным специалистам из разных

областей, включая IT, менеджмент и инженерию. Автоматизация процессов генерации кода на выбранном языке программирования из UML-диаграмм сократит время и усилия, необходимые для ручного написания кода, позволит повысить эффективность разработки и снизит вероятность ошибок, связанных с человеческим фактором. Это особенно важно в условиях современных методологий разработки программного продукта, где быстрая адаптация к изменениям и итеративный подход играют ключевую роль [1]. Поэтому разработка программного модуля для автоматической генерации программного кода на основе UML-диаграмм представляет собой актуальную задачу, способствующую оптимизации процессов разработки и повышению качества создаваемого программного продукта.

Обзор источников по теме исследования

Исследование алгоритмов, которые могут быть использованы для автоматической генерации кода (на основе шаблонов, моделей, генетических алгоритмов), и анализ их влияния на процесс разработки программного обеспечения рассмотрены в статье [2].

В качестве примеров применения различных подходов для кодогенерации можно привести следующие работы российских авторов. Так, в работе [3] представлен прототип генератора программного кода на основе диаграммы потоков данных (DFD-нотация Йордана), реализованный в виде веб-приложения для потоковой обработки данных, целевой аудиторией которого являются инженеры различных специальностей без достаточного опыта в области программирования. В работе [4] предлагается веб-ориентированное программное средство KMS, реализующее алгоритмы визуального моделирования знаний предметной области в форме диаграмм переходов состояний и деревьев событий, используемых при автоматической кодогенерации баз знаний продукционного и онтологического типа. Авторы выделяют возможность вовлечения в процесс разработки интеллектуальных

систем непрограммирующих пользователей и минимизацию ошибок программирования. В статье [5] описан конструктор кода для среды Scilab, предназначенный для автоматизации процесса создания программных модулей, программа позволяет генерировать код через интуитивно понятный пользовательский интерфейс без знания языка программирования.

Для изучения вопроса о визуальных средствах построения UML-диаграмм можно рассмотреть следующие работы. Например, в сравнительном обзоре [6] приводится анализ и сравнение инструментов для создания диаграмм, таких как Microsoft Visio, LibreOffice Draw, PlantUML и Diagrams.net (draw.io), даются рекомендации по выбору наиболее подходящего инструмента на основе индивидуальных потребностей и предпочтений пользователя. В исследовании [7] отмечается, что среди других приложений, поддерживающих создание UML-диаграмм, онлайн-редактор draw.io. отличается адаптивным пользовательским интерфейсом, а также интеграцией с сервисом хранения файлов Google.

Вопросы экспорта данных и их семантического анализа из перечисленных выше инструментов визуального моделирования UML-диаграмм в наиболее подходящие для кодогенерации форматы файлов рассмотрены в следующих работах. В статье [8] предложено решение для формирования проектной документации в формате XML при проектировании объектов, связанных с нефтегазовым комплексом. Популярный стандарт для обмена данными в Интернете JSON рассматривается в статье [9], в которой авторами предложен новый интерфейс его синтаксического анализа. Вопросы выбора стратегии синтаксического анализа для оптимизации производительности парсер-комбинаторов рассмотрены в работе [10].

Исходя из вышеизложенного, можно сделать вывод о том, что генерация кода на основе многообразных визуальных представлений проектируемого программного продукта рассматривалась в различных

аспектах современными исследователями. В качестве графического средства построения UML-диаграмм удобнее использовать популярный онлайн-редактор draw.io, позволяющий выполнять экспорт диаграмм в формат XML. Последующее преобразование в формат JSON и синтаксический анализ JSON-файла позволят сформировать требуемый программный код.

Описание методов исследования и инструментов

В качестве материалов для проведения исследования использовались UML-диаграммы, построенные в онлайн-сервисе draw.io. В частности, были рассмотрены диаграммы классов, диаграммы развертывания и диаграммы сущность-связь, представляющие собой фрагменты архитектуры программно-информационных систем, разрабатываемых студентами в рамках дисциплины «Проектирование и архитектура программных систем».

Инструментами реализации модуля для генерации кода программной системы на основе визуального представления его архитектуры послужили: язык Python в сочетании с шаблонизатором jinja2, для проверки сгенерированного кода использовался Python-модуль subprocess.run(), необходимый для компиляции Java (javac) и C++ (g++) файлов, стандартная функция Python compile() для проверки синтаксиса Python-кода, а также онлайн-редакторы для создания баз данных PostgreSQL, Oracle, MySQL.

Результаты исследования

Разработанный программный модуль поддерживает три типа UML-диаграмм. Диаграммы классов используются для создания объектно-ориентированных структур, с возможностью обеспечения различного рода статических связей, которые существуют между ними, и построения наследования, инкапсуляции и композиции. Языками для генерации кода выбраны Python, C++, Java. На основе диаграммы сущность-связь строится DDL-код для трех СУБД: PostgreSQL, Oracle, MySql. При построении должны учитываться структура и организация данных и различные

элементы, такие как таблицы их связи, имена полей, ограничения, типы данных. Диаграммы развертывания отражают инфраструктурные компоненты системы, они будут служить основой для генерации конфигурационных файлов Docker Compose.

Общая схема работы программного модуля представлена на рис.1:

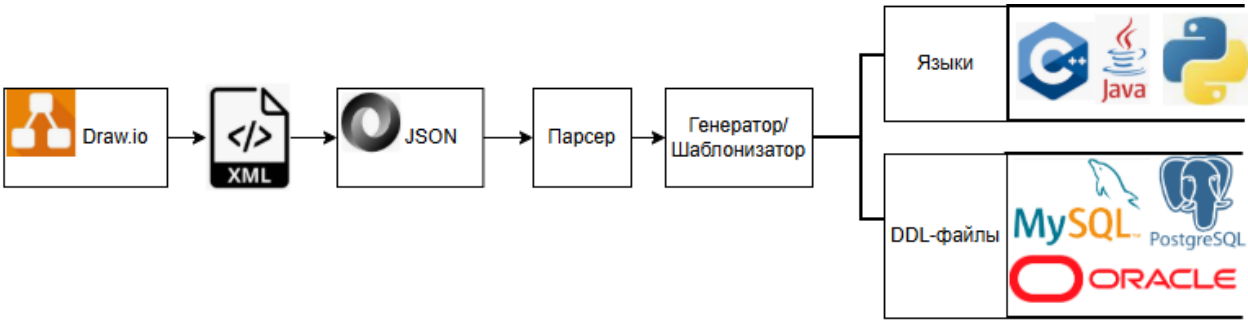

Рис. 1. – Схема работы программного модуля

Рассмотрим работу программного модуля на примере генерации кода для проекта «Система управления библиотекой».

Первоначально в онлайн-редакторе draw.io строятся диаграмма классов, диаграмма развертывания и диаграмма сущность-связь.

Диаграмма классов (рис. 2) содержит классы User (пользователь), Librarian (библиотекарь), Book (книга), Loan (сервис выдачи), SearchService (сервис поиска), Catalog (каталог); классы связаны следующими отношениями: наследование (Librarian наследуется от User), агрегация (Catalog включает Book), ассоциация (Loan связан с Book и User):

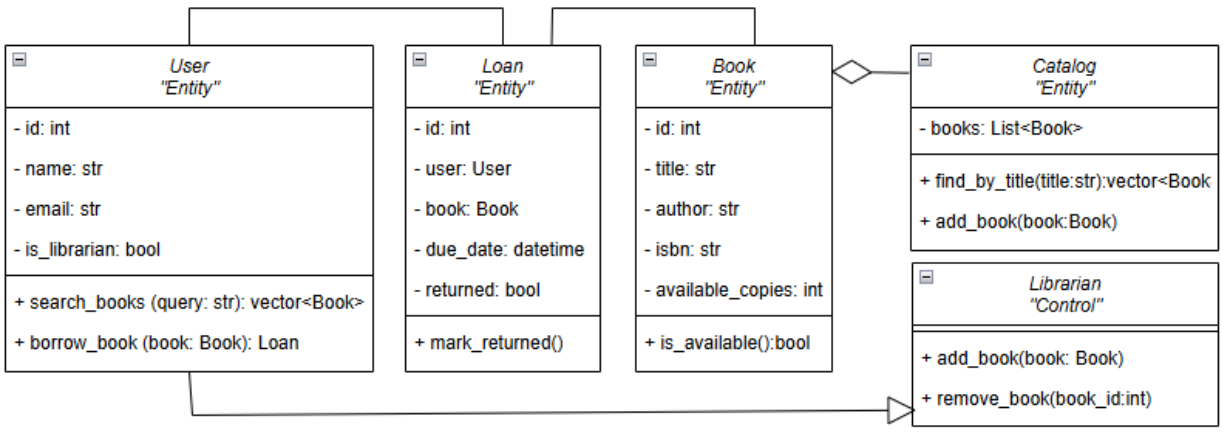

Рис. 2. – Диаграмма классов

Диаграмма сущность-связь (рис. 3) представлена таблицами Users (пользователи), Books (книги), Loans (выдачи книг), Catalog (каталог) и связями «один ко многим», реализованными через внешние ключи от Users к Loans посредством user_id, от Books к Loans посредством book_id, от Books к Catalog посредством book_id:

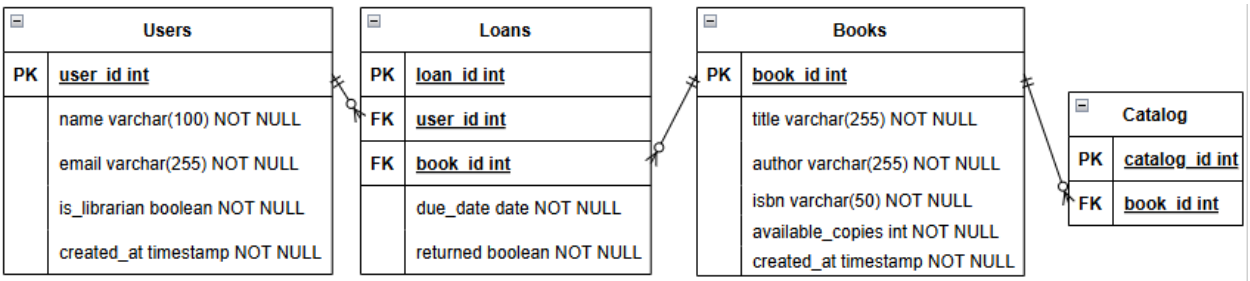

Рис. 3. – Диаграмма сущность-связь

Диаграмма развертывания (рис. 4) содержит компоненты: клиентская часть Frontend, веб-сервер и обратный прокси-сервер Nginx Reverse Proxy, серверная часть Backend, фоновый сервис, выполняющий асинхронные задачи Worker Service, высокопроизводительное хранилище ключ-значение, используемое для кэширования данных Redis Cache, реляционная база данных PostgreSQL.

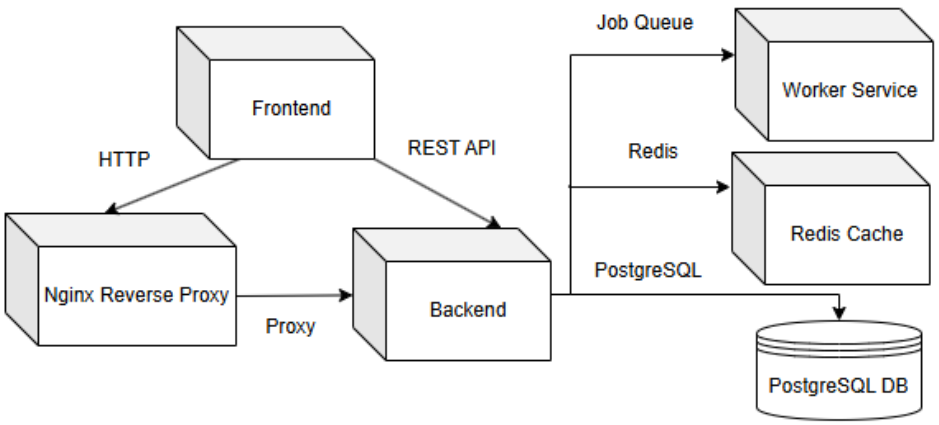

Рис. 4. – Диаграмма развертывания

Построенные диаграммы экспортируются из draw.io в XML-файлы, а затем в JSON-формат. Полученные JSON-файлы обрабатываются парсером для последующей генерации программного кода.

Рассмотрим работу парсера `parse_data()` на примере JSON-представления UML-диаграммы классов:

```
def parse_data(self, data: Dict[str, Any]) -> Dict[str, Any]:
     """ Разбирает JSON-схему UML-диаграммы классов """
     classes = data.get("classes", [])
     parsed_data = {"classes": []}
     for cls in classes:
         methods = []
         for method in cls.get("methods", []):
            params=[(param["name"], param["type"]) for param in method.get("params", [])]
            methods.append({
            "name": method["name"],
            "params": params,
            "return_type": method.get("return_type", "None")})
         parsed_data["classes"].append({
             "name": cls.get("name"),
             "attributes": [(attr["name"], attr.get("type", "Any"))
                            for attr in cls.get("attributes", [])],
             "methods": methods,
             "inherits": cls.get("inherits", None)})
     return parsed_data
```

На входе функция получает словарь *data*, в котором хранится содержимое JSON-файла. По ключу *classes* формируется список классов, каждый из которых обрабатывается отдельно: извлекаются методы класса, при этом каждый метод преобразуется в словарь с полями: *name* – имя метода, *params* – список кортежей с параметрами метода (имя и тип), *return_type* – тип возвращаемого значения (по умолчанию "None"). Для каждого класса формируется объект, содержащий следующие элементы: *name* – имя класса, *attributes* – список кортежей с атрибутами (имя и тип, *methods* – список методов, разобранных ранее, *inherits* – родительский класс (если не указан, значение None). Итоговый словарь с информацией о всех классах возвращается в переменной *parsed_data*.

После парсинга обработанные данные поступают в модуль генерации программного кода, который использует заранее подготовленные шаблоны: 3 шаблона для диаграммы классов с генерацией кода на языках C++/Java/Python; 3 шаблона для диаграммы сущность-связь с генерацией DDL-скриптов для СУБД PostgreSQL, Oracle, MySQL; 1 шаблон для генерации файла docker-compose.yaml на основе диаграммы развертывания.

Пример шаблона для СУБД MySQL в формате jinja2 приведен ниже:

```
{% for table in tables %}
CREATE TABLE `{{ table.name }}` (
     {% for column in table.columns %}`{{ column.name }}` {{ column.type }} {% if
column.primary_key %} PRIMARY KEY{% endif %} {% if column.auto_increment %}
AUTO_INCREMENT{% endif %} {% if column.not_null %} NOT NULL{% endif %} {% if not
loop.last %},
     {% endif %} {% endfor %} {% if table.columns|selectattr('foreign_key')|list %},
     {% for column in table.columns if column.foreign_key %}FOREIGN KEY (`{{
column.name }}`) REFERENCES `{{ column.foreign_key.references }}`(`{{
column.foreign_key.column }}`){% if not loop.last %},{% endif %}
     {% endfor %}
     {% endif %}
) ENGINE=InnoDB;
{% endfor %}
```

Модуль генерации работает на основе класса Generator, включающий загрузку json-файлов и проверку корректности их структуры в зависимости от типа UML-диаграммы (ClassDiagramValidator, DatabaseDiagramValidator, DockerComposeDiagramValidator), парсер json-файла, подстановку в описанные jinja2-шаблоны, валидацию полученного кода (CodeValidator). В случае успешной валидации пользователю отправляется сгенерированный код, а в случае возникновения ошибок – соответствующее уведомление.

Ниже представлены примеры сгенерированного кода для проекта «Система управления библиотекой».

Код для проекта на языке C++, сгенерированный на основе диаграммы классов (рис. 2):

```
#include <string>
#include <vector>
class User;
class Librarian;
class Book;
class Loan;
class Catalog;

class User {
public:
    std::vector<Book> search_books(std::string query);
    Loan borrow_book(Book book);
private:int id; std::string name; std::string email; bool is_librarian;};

class Librarian : public User {
public:
    void add_book(Book book); void remove_book(int book_id);
private:};
class Book {
```

```
public:
    bool is_available();
private:int id;
    std::string title; std::string author; std::string isbn; int available_copies;};

class Loan {
public:
    void mark_returned();
private:int id; User user; Book book; std::string due_date; bool returned;};

class Catalog {
public:
    std::vector<Book> find_by_title(std::string title);
    void add_book(Book book);
private:std::vector<Book> books;};
```

Стоит отметить, что описанные методы реализованы в виде прототипов, и для дальнейшего использования в проекте разработчику необходимо дописать функциональный код в тело каждого метода.

DDL-код, построенный на основе диаграммы сущность-связь, для СУБД PostgreSQL приведен ниже:

```
CREATE TABLE Users (
       user_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
       name VARCHAR(100) NOT NULL,
       email VARCHAR(255) NOT NULL,
       is_librarian BOOLEAN NOT NULL,
       created_at TIMESTAMP NOT NULL);
CREATE TABLE Books (
       book_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
       title VARCHAR(255) NOT NULL,
       author VARCHAR(255) NOT NULL,
       isbn VARCHAR(50) NOT NULL,
       available_copies INT NOT NULL,
       created_at TIMESTAMP NOT NULL);
CREATE TABLE Loans (
       loan_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
       user_id INT, FOREIGN KEY (user_id) REFERENCES Users(user_id),
       book_id INT, FOREIGN KEY (book_id) REFERENCES Books(book_id),
       due_date DATE NOT NULL,
       returned BOOLEAN NOT NULL);
CREATE TABLE Catalog (
       catalog_id INT GENERATED ALWAYS AS IDENTITY PRIMARY KEY,
       book_id INT, FOREIGN KEY (book_id) REFERENCES Books(book_id));
```

Данный код сгенерирован с соблюдением синтаксиса выбранной СУБД и не требует дополнительной доработки.

Шаблон для файла docker-compose.yaml содержит структурные элементы (Frontend, Backend, Nginx Reverse Proxy, Worker Service, Redis

Cache, PostgreSQL), соответствующие диаграмме развертывания (рис. 4). Фрагмент данного шаблона представлен ниже:

```
version: "3.9"
services:
 frontend:
      container_name: frontend
      image: image_registry_url                    # Manually(Optional if build)
      build:     context: ./path                   # Manually
              dockerfile: ./path/.../Dockerfile    # Manually
      environment: - ENV_NAME:ENV_VALUE            # Manually
      ports: - "1111:1111"                         # Manually
      volumes: - ./path:/app                       # Manually
 backend:
      container_name: backend
      image: image_registry_url                    # Manually(Optional if build)
      build:     context: ./path                   # Manually
              dockerfile: ./path/.../Dockerfile    # Manually
      environment: - ENV_NAME:ENV_VALUE            # Manually
      ports:  - "1111:1111"                        # Manually
      volumes:  - ./path:/app                      # Manually
      depends_on:
        - frontend
        - nginx-reverse-proxy
```

Для использования данного кода необходимо дописать правильные значения ключей в строках, отмеченных комментарием #Manually.

Для тестирования представленного модуля был использован 41 проект программно-информационных систем различного назначения, разрабатываемых студентами направления подготовки 09.03.04 Программная инженерия по дисциплине «Проектирование и архитектура программных систем». Следует отметить, что проекты имели однотипный набор функциональных возможностей (авторизация/регистрация пользователей, сортировка и фильтрация данных, предоставление личного кабинета пользователя, реализация административной панели, оформление заказов, формирование отчетов). Результаты тестов показали, что сгенерированный с помощью разработанного модуля код полностью соответствует исходным UML-диаграммам, включая структуру классов, связи между ними, а также конфигурацию базы данных и инфраструктуры (Docker Compose).

Заключение

Одним из важных инструментов для визуализации архитектуры проекта по созданию программного обеспечения являются UML-диаграммы. Перевод построенных диаграмм в программный код – это длительный и сложный процесс, требующий участия высококвалифицированных программистов, поэтому проблему автоматизации кодогенерации можно считать актуальной проблемой.

В ходе проведённого исследования был разработан модуль для автоматической генерации кода на основе UML-диаграмм. В частности, 1) были успешно сгенерированы программные классы на языке Python, отражающие структуру классов, отношения наследования, агрегации и ассоциации в соответствии с UML-диаграммой классов; 2) созданы корректные DDL-скрипты для СУБД PostgreSQL, сохранены все внешние ключи, типы данных адаптированы под особенности СУБД.; 3) сформирован базовый шаблон docker-compose.yaml, охватывающий веб-клиент, backend, базу данных, Redis и другие компоненты, согласно UML-диаграмме развертывания; 4) установлено полное соответствие между логикой UML-диаграмм и структурой сгенерированных компонентов на всех уровнях: объектном, реляционном и архитектурном.

Проведенное тестирование подтверждает, что автоматизация написания кода позволяет значительно сократить время на начальную разработку и при этом получить работоспособный исходный код.

Разработанный модуль предоставляет необходимый функционал для повышения эффективности разработки ПО и снижения вероятности ошибок, связанных с человеческим фактором, может быть применим как в образовательных целях, так и в профессиональной деятельности.

В качестве возможных перспектив развития проекта следует выделить расширение списка поддерживаемых языков программирования и СУБД, а также интеграцию с популярными инструментами моделирования.

Литература

1. Манжикова С. Ц. UML как инструмент управления программными проектами // Вестник Кыргызско-Российского Славянского университета. 2023. Т. 23. № 12. С. 88-95. DOI: 10.36979/1694-500X-2023-23-12-88-95.

2. Косырев Е.А. Исследование алгоритмов для автоматической генерации кода для ускорения процесса разработки программного обеспечения // Современные научные исследования и инновации. 2023. №10. URL: web.snauka.ru/issues/2023/10/100884.

3. Минакова О.В., Трубников И.В., Курипта О.В. Построение генератора программного кода для решения инженерных задач // Вестник воронежского государственного технического университета. 2020. №3. С. 14-19.

4. Юрин А. Ю., Дородных Н. О. Программная реализация алгоритмов для создания прототипов баз знаний на основе визуального моделирования и трансформаций // Программные продукты и системы. 2024. № 3. С. 324-333. DOI: 10.15827/0236-235X.147.324-333.

5. Лазарева Н. Б., Булканов Д. Е. Конструктор кода для среды Scilab // Инженерный вестник Дона. 2025. № 1. URL: ivdon.ru/ru/magazine/archive/n1y2025/9790.

6. Машина А. А., Кузина В. В. Инструменты для создания диаграмм: сравнение и выбор // Образование и наука в современном мире. Инновации. 2024. № 3. С. 245-251.

7. Marthiawati N., Kurniawansyah K., Nugraha H., Khairunnisa F. Pelatihan Pembuatan UML (Unified Modelling Language) Menggunakan Aplikasi Draw.io Pada Prodi Sistem Informasi Universitas Muhammadiyah Jambi // Transformasi

Masyarakat: Jurnal Inovasi Sosial dan Pengabdian. 2024. Vol. 1. No. 2. pp. 25-33. DOI: 10.62383/transformasi.v1i2.109.

8. Денисов В. И., Луценко О. Н., Лапковская В. В., Тепляков Н. М. Новый этап развития цифровизации документооборота – работа с проектной документацией в формате XML // Известия Томского политехнического университета. Промышленная кибернетика. 2023. Т. 1. № 1. С. 17-23.

9. Keiser J., Lemire D. On-demand JSON: A better way to parse documents? // Software: Practice and Experience. 2024. №6. pp. 1074-1086.

10. Чубейко С.В., Цуриков А.Н., Палагута В.С. Анализ синтаксического разбора текста с помощью парсер-комбинаторов // Инженерный вестник Дона. 2018. №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5335.

References

1. Manzhikova S. Cz. Vestnik Ky`rgy`zsko-Rossijskogo Slavyanskogo universiteta. 2023. Vol. 23. № 12. pp. 88-95. DOI: 10.36979/1694-500X-2023-23-12-88-95.

2. Kosy`rev E.A. Sovremenny`e nauchny`e issledovaniya i innovacii. 2023. №10. URL: web.snauka.ru/issues/2023/10/100884.

3. Minakova O.V., Trubnikov I.V., Kuripta O.V. Vestnik voronezhskogo gosudarstvennogo texnicheskogo universiteta. 2020. № 3. pp. 14-19.

4. Yurin A. Yu., Dorodny`x N. O. Programmny`e produkty` i sistemy` (Rus). 2024. № 3. pp. 324-333. DOI: 10.15827/0236-235X.147.324-333.

5. Lazareva N. B., Bulkanov D. E. Inzhenernyj vestnik Dona, 2025, № 1. URL: ivdon.ru/ru/magazine/archive/n1y2025/9790.

6. Mashina A. A., Kuzina V. V. Obrazovanie i nauka v sovremennom mire. Innovacii. 2024. № 3. pp. 245-251.

7. Marthiawati N., Kurniawansyah K., Nugraha H., Khairunnisa F. Transformasi Masyarakat : Jurnal Inovasi Sosial dan Pengabdian. 2024. Vol. 1. No. 2. pp. 25-33. DOI: 10.62383/transformasi.v1i2.109.

8. Denisov V. I., Lucenko O. N., Lapkovskaya V. V., Teplyakov N. M. Izvestiya Tomskogo politexnicheskogo universiteta. Promy`shlennaya kibernetika p. 2023. Vol. 1. № 1. pp. 17-23.

9. Keiser J., Lemire D. Software: Practice and Experience. 2024. №6. pp. 1074-1086.

10. Chubejko S.V., Czurikov A.N., Palaguta V.S. Inzhenernyj vestnik Dona, 2018, №4. URL: ivdon.ru/ru/magazine/archive/n4y2018/5335.

Дата поступления: 6.06.2025

Дата публикации: 26.07.2025