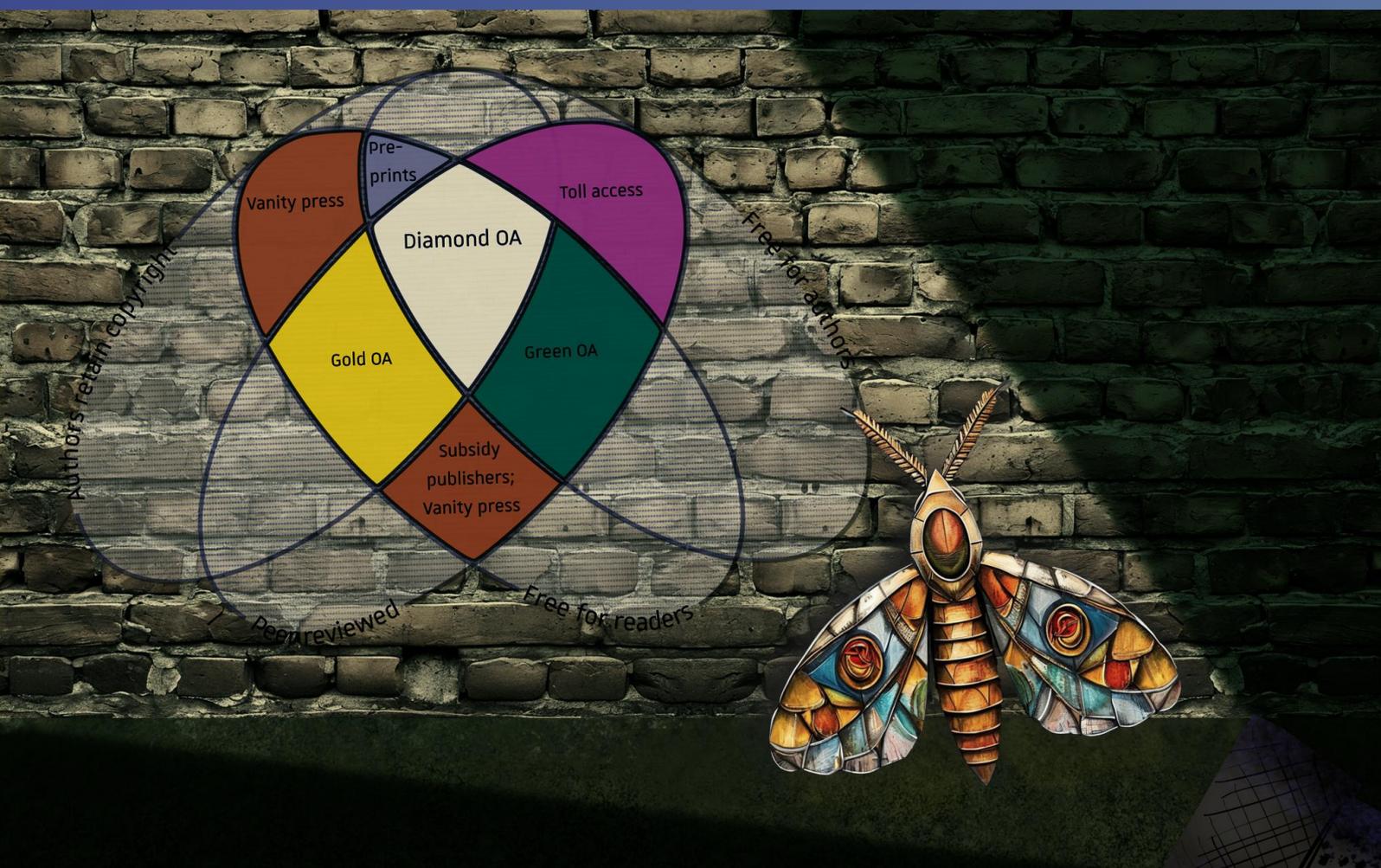


ЦИФРОВАЯ ЭКОНОМИКА



Редакционный совет электронного журнала «Цифровая экономика»

- Агеев Александр Иванович – д.э.н., генеральный директор Института экономических стратегий, заведующий кафедрой НИЯУ «МИФИ», профессор, академик РАН.
- Афанасьев Михаил Юрьевич – д.э.н. Заведующий лабораторией прикладной эконометрики ЦЭМИ РАН
- Бабаян Евгений Борисович – Генеральный директор НП «Агентство научных и деловых коммуникаций»
- Бахтизин Альберт Рауфович – член-корреспондент РАН, д.э.н., профессор РАН, директор ЦЭМИ РАН
- Войниканис Елена Анатольевна – д.ю.н. Ведущий научный сотрудник Института права и развития ВШЭ — Сколково.
- Волынкина Марина Владимировна – д.ю.н. Ректор НОЧУ ВПО «Институт гуманитарного образования и информационных технологий.
- Гурдус Александр Оскарович – д.э.н., к.т.н., президент группы компаний «21Company».
- Димитров Илия Димитрович – исполнительный директор НКО «Ассоциации Электронных Торговых Площадок».
- Ерешко Феликс Иванович – д.т.н. профессор, заведующий отделом информационно-вычислительных систем (ИВС) ВЦ РАН.
- Засурский Иван Иванович – к.ф.н. президент Ассоциации интернет-издателей, заведующий кафедрой новых медиа и теории коммуникации факультета журналистики МГУ имени М.В. Ломоносова
- Калятин Виталий Олегович – к.ю.н., профессор Исследовательского центра частного права при Президенте РФ им. С.С. Алексеева
- Китова О.В. – д.э.н., к.ф.-м.н. зав. кафедрой Информатики РЭУ им. Г.В. Плеханова.
- Козырь Юрий Васильевич – д.э.н., главный научный сотрудник ЦЭМИ РАН
- Ливадный Евгений Александрович – к.т.н., к.ю.н., Руководитель проектов по интеллектуальной собственности Государственной корпорации «Ростех».
- Макаров Валерий Леонидович – академик РАН, научный руководитель ЦЭМИ РАН
- Паринов Сергей Иванович – д.т.н., главный научный сотрудник ЦЭМИ РАН.
- Райков Александр Николаевич – д.т.н., профессор, ведущий научный сотрудник Института проблем управления РАН, Генеральный директор ООО «Агентство новых стратегий»
- Семячкин Дмитрий Александрович – к.ф.-м.н., директор Ассоциации «Открытая наука»
- Серго Антон Геннадьевич – д.ю.н., Профессор кафедры авторского права, смежных прав и частоправовых дисциплин Российской государственной академии интеллектуальной собственности (РГАИС)
- Соловьев Владимир Игоревич – д.э.н. руководитель департамента анализа данных, принятия решений и финансовых технологий Финансового университета при Правительстве РФ
- Фролов Владимир Николаевич, – д.э.н., профессор, научный руководитель проекта «Copernicus Gold».
- Хохлов Юрий Евгеньевич – к.ф.-м.н., доцент, председатель Совета директоров Института развития информационного общества, академик Российской инженерной академии
- Терелянский Павел Васильевич, – д.э.н., профессор, главный научный сотрудник Научно-исследовательского института "Управления цифровой трансформацией экономики", ФГБОУ ВО "Государственный университет управления".

Миссия журнала

Миссия журнала — поддерживать высокий научный уровень дискуссии о цифровой экономике, методах ее изучения и развития, вовлекая в этот процесс наиболее квалифицированных экспертов – исследователей и практиков; доносить научное знание о самых сложных ее аспектах до тех, кто реально принимает решения, и тех, кто их исполняет. Одновременно журнал направлен на обеспечение возможности для обмена мнениями между профессиональными исследователями.

Название и формат издания

Название «Цифровая экономика» подчеркивает междисциплинарный характер журнала, а также ориентацию на новые методы исследования и новые формы подачи материала, возникшие вместе с цифровой экономикой. В современном ее понимании цифровая экономика – не только новый сектор экономики, но и новые методы сбора информации на основе цифровых технологий, психометрия и компьютерное моделирование, а также иные методы экспериментальной экономики.

Тематика научных и научно-популярных статей

Основную тематику журнала представляют научные и научно-популярные статьи, находящиеся в предметной области цифровой экономики, информационной экономики, экономики знаний. Основное направление журнала – это статьи, освещающие применение подходов и методов естественных наук, математических моделей, теории игр и информационных технологий, а также использующие результаты и методы естественных наук, в том числе, биологии, антропологии, социологии, психологии.

В журнале также публикуются статьи о цифровой экономике и на связанные с ней темы, в том числе, доступные для понимания людей, не изучающих предметную область и применяемые методы исследования на профессиональном уровне. Основная тема – создание и развитие единого экономического пространства России и стран АТР. Сюда можно отнести статьи по обсуждаемым вопросам оптимизации использования ресурсов и государственному регулированию, по стандартам в цифровой экономике. Сегодня или очень скоро это стандарты – умный город, умный дом, умный транспорт, интернет вещей, цифровые платформы, BIM-технологии, умные рынки, умные контракты, краудсорсинг и краудфандинг и многие другие.

Журнал «Цифровая экономика», № 1(31) 2025

Выпуск № 1, 2025 год

Журнал зарегистрирован Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций

Свидетельство о регистрации № ЭЛ № ФС77-70455 от 20 июля 2017 г.

Редакционная коллегия

Козырев А. Н. – главный редактор, д.э.н., к.ф.-м.н., руководитель научного направления – математическое моделирование, г.н.с. ЦЭМИ РАН

Ведута Е. Н. – д.э.н., профессор, зав. кафедрой стратегического планирования и экономической политики факультета государственного управления имени М. В. Ломоносова

Гатауллин Т.М. – д.э.н., к.ф.-м.н., зам. директора Центра цифровой экономики Государственного университета управления

Китов Владимир Анатольевич, к.т.н., зам. Зав. кафедрой Информатики по научной работе РЭУ им. Г.В. Плеханова

Костин А.В. – к.э.н., ведущий научный сотрудник ЦЭМИ РАН

Лугачев М.И. – д.э.н., заведующий кафедрой Экономической информатики Экономического факультета МГУ имени М.В. Ломоносова.

Макаров С.В. – к.э.н., ведущий научный сотрудник ЦЭМИ РАН.

Неволин И.В. – к.э.н., ведущий научный сотрудник ЦЭМИ РАН

Ноак Н.В. – к.п.н., ведущий научный сотрудник ЦЭМИ РАН

Скрипкин К.Г. – к.э.н., доцент кафедры Экономической информатики Экономического факультета МГУ имени М.В. Ломоносова.

Тевелева О.В. – к.э.н., старший научный сотрудник ЦЭМИ РАН

Писарева О.М. – к.э.н., заведующий кафедрой математических методов в экономике и управлении, Директор Института информационных систем ФГБОУ ВО "Государственный университет управления" (ГУУ)

Чесноков А.Н. – руководитель проекта АН2

Все работы опубликованы в авторской редакции.

Композиция на обложке составлена Елизаветой Вершининой.

Подписано к опубликованию в Интернете 10.03.2025, Авт. печ. л. 9,7

Сайт размещения публикаций: <http://digital-economy.ru/>

Адрес редакции: 117418 Москва, Нахимовский проспект, 47, комн. 520

При использовании материалов ссылка на журнал «Цифровая экономика» и на автора статьи обязательна (на условиях creative commons).

© Журнал «Цифровая экономика», 2025

I S S N 2 6 8 6 - 9 5 6 X



9 772686 956001 >

СОДЕРЖАНИЕ

Слово редактора.....	4
1. Научные статьи.....	5
1.1. Трегубов В.Н., Матушкин М.А., Использование технологии генеративного искусственного интеллекта для проектирования цифровых двойников в логистике.....	5
1.2. Зекирьяев Р.Т., Болбаков Р.Г. Практическая реализация методологии CODE	14
1.3. Недоря А.Е., Разработка языка Тривиль. Часть 3. Баланс	24
1.4. Недоря А.Е., Разработка языка Тривиль. Часть 4. Реализация	38
1.5. Никитчук С.С., Китова О.В., Цифровая трансформация торгово-промышленных предприятий на основе использования инновационных практик.....	46
1.6. Терелянский П.В., Как искусственный интеллект разрушит экономику и уничтожит человечество	53
1.7. Афанасьев А.А., Пономарева О.С., Исполнение уточненного эконометрического прогноза на 2024 г. ВВП России в условиях специальной военной операции	63
1.8. Ратман Л.П., Использование нейросетей в маркетинговой деятельности строительной компании.....	69
1.9. Грачёв И.Д., Ларин С.Н., Ноак Н.В., Цифровое моделирование НДС и налога с оборота	74
2. Обзоры	83
2.1. Китов В.А., Китова О.В., Советские истоки компьютерной эры в Китае	83
3. Мнения.....	88
3.1. Фролов В.Н., Романчук А.В., Дорофеев А.В. Банк-как-Сервис – параллельная реальность финансовых экосистем.....	88

Слово редактора

Дорогие читатели, перед вами тридцать первый с начала выпуска и первый в 2025 году номер журнала «Цифровая экономика». Мы стараемся не допустить задержки с публикацией научных текстов, оформленных в соответствии с нашими требованиями и успешно прошедших рецензирование. К сожалению, авторы слишком часто нарушают эти требования, а у нас не всегда хватает терпения и сил, чтобы привести статьи в нужный формат. Вместе с тем иногда возникают обстоятельства, требующие ускорить выпуск очередного номера. А потому хочу еще раз напомнить: от авторов требуется готовить тексты сразу в том формате, в каком они могут быть опубликованы.

Авторам настоятельно рекомендуется внимательно читать памятку, публикуемую на последней странице каждого выпуска журнала, и строго следовать рекомендациям. Это снимает часть забот и технической работы с команды, выпускающей журнал на общественных началах. В стандарте, получившем (независимо) названия brilliant и platinum. Можно относиться к этому с юмором, но помнить о последствиях. Статьи, подготовленные в точном соответствии с требованиями, пользуются приоритетом при рассмотрении вопроса о включении в очередной выпуск. В результате статьи, подготовленные без учета наших требований, продолжают ожидать официальной публикации с присвоением doi и других метаданных, от чего теряют только авторы, поскольку читателям их тексты доступны.

Основная тематика данного выпуска – работа с данными и искусственный интеллект, что связано с актуальностью этой тематики, включая особое внимание к ней со стороны высшего руководства России. Актуальность тематики нашла отражение и в статьях, присылаемых в журнал для публикации. Большая часть из них так или иначе касается искусственного интеллекта. Эта тема затрагивается в разных статьях с точки зрения математики, техники, экономики и приложений в разных областях.

От предыдущих нескольких номеров этот номер отличается тем, что, помимо раздела «Научные статьи», в нем снова появились разделы «Обзоры» и «Мнения», что обусловлено определенными событиями в медиасфере, связанными с новыми разработками в мире нейросетей и финансов.

Раздел «Обзоры» включает одну статью, написанную Владимиром Анатольевичем Китовым и Ольгой Викторовной Китовой, принимавшими самое непосредственное участие и в создании нашего журнала, и во многих событиях в мире вычислительной техники, ставших историей. Статья посвящена истории помощи советских ученых, прежде всего, Анатолия Ивановича Китова, Исаака Семёновича Брука, Сергея Алексеевича Лебедева и Николая Андреевича Криницкого, китайским коллегам в становлении компьютерного производства. В статье прослеживается связь первых достижений китайских ученых в области информатики с историей создания вычислительной техники в СССР.

Её актуальность обусловлена тем, что недавние исследования в области истории вычислительной техники, опубликованные на английском языке западными историками, могут создать у ученых впечатление, что влияние советских ученых на китайских коллег на начальном этапе компьютерной эры в Китае было весьма незначительным. Но это далеко не так. Поскольку одна из миссий журнала – восстановление правдивой картины развития информатики как науки в нашей стране, мы публикуем краткое изложение результатов основанного на фактах (а не домыслах) исследования на эту тему.

Исследование основано на анализе научных работ по истории вычислительной техники, материалах Виртуального компьютерного музея, фондах Государственного политехнического музея России, архивах Российского экономического университета им. Г.В. Плеханова, а также на личных встречах авторов с китайскими учеными и переводчиками.

Раздел «Мнения» также состоит из одной статьи на очень серьезную тему, подготовленной тремя специалистами, непосредственно погруженными в её тематику как практики. В статье рассмотрена новейшая технология Банк-как-Сервис (Banking-as-a-Service, принятое сокращение BaaS) в финансовой индустрии, история платёжных решений, предпосылки появления BaaS, особенности применения, преимущества использования этого решения для банков и небанковских компаний, а также вызовы для консервативного банковского сообщества. Авторы полагают, что она написана просто, понятно и конструктивно. Будет полезна и интересна как для профессионалов, так и для людей, далёких от банковского бизнеса. Этот текст чрезвычайно важен для тех, кто хотел бы знать, куда стремительно мчится мировая финансовая система. А она закладывает немыслимый вираж, и на обочине окажутся многие не успевающие за инновациями, очень быстро меняющимися финансовый ландшафт.

Всем потенциальным читателям желаю, как всегда, легкого чтения.

Главный редактор журнала

д.э.н. А. Н. Козырев

1. НАУЧНЫЕ СТАТЬИ

УДК: 004.82, 621.13

1.1. Использование технологии генеративного искусственного интеллекта для проектирования цифровых двойников в логистике

Трегубов В.Н., Матушкин М.А.,
ФГБОУ ВО Саратовский государственный технический университет имени Гагарина Ю.А.,
г. Саратов, Россия

Аннотация: Авторами исследуется концепция «Логистика 4.0», которая формируется на принципах «Индустрии 4.0» и направлена на цифровую трансформацию логистических процессов с использованием искусственного интеллекта и других цифровых технологий. Использование в логистике интернета вещей (IoT), киберфизических систем, больших данных и облачных вычислений позволяет создавать цифровые двойники логистических систем. Это способствует повышению экономической, социальной и экологической устойчивости, обеспечивая прозрачность, безопасность и оперативность управления цепочками поставок. Эффективным инструментом моделирования для создания цифровых двойников является унифицированный язык моделирования (UML). Российские и зарубежные исследования демонстрируют его хорошую применимость для описания архитектуры, функциональности и поведения логистических систем. Интеграция UML с другими методами, например моделью спецификации дискретных событий, расширяет его возможности как инструмента концептуального моделирования. В статье рассматривается применение генеративного искусственного интеллекта (ГИИ) для создания диаграмм прецедентов при разработке цифрового двойника логистической системы. Хотя предложенное ГИИ решение не является идеальным, оно демонстрирует потенциал ГИИ в решении подобных задач, и созданная модель может служить отправной точкой для дальнейшей разработки информационной системы. В будущем интеграция ГИИ в программные средства для создания информационных систем откроет новые возможности формирования логистических процессов.

Введение

Использование искусственного интеллекта (ИИ) в логистике представляет собой перспективное направление развития, которое может быть реализовано в рамках реализации концепции «Логистика 4.0», которая формирует единую методологию цифровизации в логистике и включает в себя различные цифровые технологии, такие как интернет вещей (IoT), киберфизические системы, большие данные, облачные вычисления, мобильные системы и даже социальные сети. Логистическая система, построенная на принципах «Логистика 4.0», позволяет удовлетворять индивидуальные запросы клиентов без увеличения их затрат и обеспечивает эффективную цифровизацию логистики в том числе и с помощью методов ИИ.

«Логистика 4.0» ориентирована на создание цифровых двойников, когда виртуальная модель цепи поставок выполнена, например в виде симуляции, непрерывно связана с реальной логистической системой. Такой подход формирует обмен информацией между реальным миром и виртуальным пространством. Это обеспечивает повышение устойчивости цепи поставок, способствуют улучшению операционной эффективности и наглядности представления логистических процессов. С помощью технологии цифровых двойников обеспечивается рост экономической эффективности цепи поставок и повышение ее экологической устойчивости. Важным элементом технологии является возможность отслеживать и контролировать перемещение грузов в режиме реального времени, что в современных условиях при снижении стоимости датчиков отслеживания можно выполнять даже на уровне отдельных товарных единиц.

Для моделирования цифровых двойников нами предлагается использовать унифицированный язык моделирования (UML). Это специализированный язык, который широко используется для визуального представления процессов и поведения программных систем. Он применяется для моделирования программных, бизнес-систем, основанных на объектно-ориентированном подходе. Преимущества UML включают в себя: гибкость, удобочитаемость, точное представление архитектуры системы, упрощение планирования и отладки программ. UML широко используется в исследовательских работах для создания прототипов систем.

В нашем исследовании мы продемонстрируем, как с использованием генеративного искусственного интеллекта и UML может быть спроектирована система цифровых двойников в виде UML диаграмм вариантов использования.

Теоретический обзор

Широкое использование искусственного интеллекта возможно в рамках формирования концепции «Логистика 4.0» по аналогии с концепцией «Индустрия 4.0», применяемой в промышленном производстве. В ее основе лежит возможность использования цифровых технологий при выполнении логистических операций [1]. Изначально «Индустрия 4.0» представляла собой инициативу, которая была направлена на использование цифровых технологий для повышения конкурентоспособности отраслей тяжелой промышленности. Она была предложена и первоначально разработана немецкими исследователями, а ее ключевая идея заключалась в применении интеллектуальных технологий для автоматизации производств, роботизации, формирования киберфизических систем и использования интеллектуальных технологий для повышения конкурентоспособности промышленных компаний.

В дальнейшем «Индустрия 4.0» стала распространяться на другие сферы, а предлагаемая метафора использовалась все более широко, появились различные варианты систем 4.0, в частности «Логистика 4.0», «Снабжение 4.0», «Закупки 4.0». Все они используются для цифровизации различных сфер логистики и включают в себя вспомогательные технологии, которые адаптируются под потребности клиентов и помогают решать их проблемы, например, достижение устойчивости цепи поставок (Evtodieva и др. 2019), повышение эффективности логистического процесса (Amr, Ezzat, и Kassem 2019), сбор больших данных о логистических процессах для их дальнейшего использования в моделировании (Khan и др. 2022).

С учетом специфики логистики, особенно в связи с тем, что цифровых технологий в ней широко использовались всегда, ряд авторов задается вопросом, не является ли «Логистика 4.0» просто модным словом, а не новым вкладом в развитие логистической технологии (Szymańska 2017). Для разрешения этого противоречия сформировалось понимание, что концепция «Логистика 4.0» должна включать в себя ряд вспомогательных технологий, которые помогают эффективно решать логистические проблемы. В обзоре литературы (Winkelhaus и Grosse 2019) авторами выделены ключевые составляющие «Логистика 4.0». Среди них: интернет вещей (IoT), киберфизические системы, большие данные, облачные вычисления, мобильные системы, системы на основе социальных сетей, большие данные, интеллектуальный анализ данных, облачные вычисления, искусственный интеллект. Во время пандемии COVID-19 именно переход к «Логистика 4.0», включая IoT, искусственный интеллект, блокчейн, облачные вычисления и дополненную реальность, рассматривалась как фактор повышения устойчивости цепочек поставок (Gupta и др. 2022).

Одним из инструментов «Логистика 4.0» является создание цифровых двойников, когда виртуальная модель цепи поставок, выполненная в виде симуляции, постоянно связана с реальной логистической системой. Такой подход формирует прямую связь между реальным миром (фактическими событиями в цепи поставок) и виртуальным пространством - имитационной моделью. Для реализации этого подхода используются технические устройства, которые позволяют отслеживать и контролировать перемещение грузов в режиме реального времени, что максимально способствует цифровой трансформации сферы логистики. Результаты исследований показывают, что внедрение технологий отслеживания в цепях поставок позволяет повысить эффективность логистики, обеспечить высокую прозрачность логистических процессов и безопасность транспортировки грузов. Экономическая эффективность внедрения таких технологий зависит от объема операций, стоимости доставки грузов, возможности повторного использования меток отслеживания, продолжительности поездок и структуры цепочки поставок (Le и Fan 2024).

Первоначально устройства для отслеживания устанавливались только на автомобили, морские суда или контейнеры. В настоящее время, за счет существенного снижения стоимости устройств слежения, они могут использоваться гораздо шире, например, ими могут оснащаться отдельные товарные единицы с дорогостоящими товарами, а также отдельные паллеты или товарные сборки. Развитие облачных технологий помогает создавать централизованные хранилища для агрегации данных, полученных в результате отслеживания, а также создавать гибридные системы, позволяющие объединять информацию об отслеживании с традиционными системами управления (Liu, Pan, и Ballot 2024).

Кроме того, «Логистика 4.0» обеспечивает повышение устойчивости организации, примеры, приведенные в исследовании (Hrouga и Sbihi 2023) убедительно показывают, что повышение операционной эффективности и наглядности с помощью цифровых технологий обеспечивает повышение производительности и эффективности цепи поставок и их социальной и экологической устойчивости.

В нашем исследовании мы будем опираться на понятие «Логистики 4.0», которое основано на определении из (Winkelhaus и Grosse 2019): «Логистика 4.0 - логистическая система, позволяющая устойчиво удовлетворять индивидуальные запросы клиентов без увеличения затрат и поддерживающая это развитие в промышленности и торговле с помощью ее цифровизации и использование технологий искусственного интеллекта». При таком подходе не нужно полагаться на определённые вспомогательные технологии, а необходимо сосредоточиться на потребностях в удовлетворении запросов клиентов, которые реализуются технологиями цифровизации.

Для проектирования информационных систем в рамках концепции «логистика 4.0» можно использовать унифицированный язык моделирования (UML), который станет инструментом для наглядного представления процессов и поведения системы. Он позволяет выполнить стандартизацию визуального представления информационных систем (Галиаскаров Э.Г. 2024). UML широко применяется для моделирования как программных, так и для бизнес-систем. В основе UML лежит объектно-ориентированный

подход, который подразумевает разложение сложной системы на естественно существующие объекты, содержащие данные и процессы взаимодействия этих объектов. С помощью UML появляется возможность визуализировать сложные системы как в статическом, так и в динамическом аспектах.

Использование UML в моделирование логистических систем имеет достаточно длительную историю. Анализ, проведенный нами по публикациям российских исследователей, показывает, что UML использовался в логистических системах различного вида. Например, в статье (Ивашенко 2006) авторами рассматриваются ключевые аспекты создания логистических систем для пассажирского транспорта с применением унифицированного языка моделирования. В работе представлены основные концепции и схемы языка моделирования, а также их практическое применение. В статье приведены примеры создания схем, описывающих работу логистической системы пассажирского транспорта.

В исследовании (Абд Алхади 2023) авторами были проанализированы ключевые объектно-ориентированные модели информационной системы транспортной логистики. Модели были разработаны с использованием языка UML и включают в себя различные типы диаграмм: вариантов использования, классов, последовательности и компонентов. Диаграммы были созданы на основе ранее разработанных математических моделей принятия решений сотрудниками транспортно-логистического предприятия.

В зарубежных исследованиях больше внимания уделяется прикладному использованию этой технологии, например для описания методов обеспечения безопасности информационной системы. Используются UML-стереотипы которые помогают разработчикам выявлять уязвимые участки логистической цепи и автоматизировать разработку тестовых примеров безопасности (Dakis и др. 2018). В исследовании (Zhiyong, Jie, и Xinhui 2013) представлена модель спецификации дискретных событий (DEVS), которая применяется в логистической системе вместе с моделированием UML. Авторы считают, что система DEVS является отличным инструментом для концептуального моделирования и предлагают инструменты для улучшения модели, расширив ее с помощью моделирования UML. В статье (Yonghua Zhou, Yuliu Chen, и Huaru Lu 2004) авторы описывают, как UML был использован в рамках пилотного проекта в Китае. Технологические и информационные объекты моделировались с помощью диаграмм вариантов использования, диаграмм последовательности и диаграммы классов.

Обобщая данные исследования заметим, что все авторы указывают, что преимущества моделирования с помощью UML заключаются в его гибкости, удобочитаемости, способности точно представлять архитектуру системы и упростить процесс планирования и разработки программ на конкретных языках реализации. Многие исследовательские работы используют моделирование UML для демонстрации различных систем, охватывающих разнообразные области, функциональные возможности, структуру и поведение.

Проектирование информационной системы цифровых двойников

Разработка UML модели UML с использованием текстового формата

Для создания диаграмм UML различных типов традиционно использовались специализированные графические редакторы, например Microsoft Visio или Draw.IO, в которых представлены графические примитивы с основными элементами UML. В последние годы большое распространение получили редакторы, которые позволяют создавать диаграммы UML на основе их текстового описания. К таким инструментам относятся Mermaid и PlantUML (PlantUML 2025).

PlantUML – это открытый формат описания диаграмм и кроссплатформенный редактор для построения диаграмм UML на основе их текстового описания. Помимо диаграмм UML с помощью редактора PlantUML можно создавать и некоторые другие виды диаграмм, например диаграммы Ганта и интеллектуальные карты. PlantUML это удобный и эффективный онлайн редактор, который позволяет быстро подготовить иллюстрации для программной документации в виде диаграмм UML.

Диаграммы UML создаются с помощью специализированного языка описания диаграмм PlantUML. Язык PlantUML поддерживает создание различных диаграмм, имеет развитую систему команд и поддерживает большой набор элементов. Изначально язык PlantUML был разработан только для создания диаграмм UML, однако затем был существенно расширен разработчиками и в настоящее время позволяет создавать широкий спектр диаграмм. Язык PlantUML поддерживают различные онлайн-редакторы, которые позволяют создавать UML диаграммы без установки специализированного программного обеспечения на компьютер. Официальный редактор расположен на сайте PlantUML.com. Однако, существуют более эффективные с точки зрения реализации интерфейса варианты, например редактор PlantText (PlantText 2025) Данный редактор предлагает удобный пользовательский интерфейс для ввода кода, шаблоны и подсказки, а также подсветку синтаксиса и автозавершение ввода команд.

Подобно языкам программирования PlantUML поддерживает возможность подключения дополнительных библиотек, предлагающих функциональность, отсутствующую в ядре языка. Например, с использованием дополнительной библиотеки на диаграмму можно добавить пиктограммы и использовать расширенные способы разметки, которые позволяют добавить пиктограммы на диаграмму PlantUML. Стандартная библиотека встроена в базовый PlantUML, поэтому для ее использования не требуется ничего дополнительно загружать.

Готовая диаграмма может быть экспортирована в различных в графических форматах. Если текстовая документация создается в специализированных системах, которые поддерживают формат

Markdown, то может быть использован механизм прямого встраивания кода PlantUML в текстовом формате. Такую возможность представляет, например, система контроля версий GitLab, которая позволяет включить текст в формате PlantUML прямо в документ и при отображении такого документа текст автоматически преобразуется в изображения. При необходимости в диаграмму можно внести правки, которые изменят результат.

Генерация моделей с использованием искусственного интеллекта

Одним из способов применения генеративного искусственного интеллекта (ГИИ) в практических целях является визуализация представленных данных в виде специализированных диаграмм, а также описание бизнес-процессов и логистических систем. Мы определили несколько областей применения технологий ГИИ для создания визуализации логистических процессов (Трегубов 2024):

- Создание блок-схем логистических процессов на основе их текстового описания и библиотеки Mermaid.
- Структурированное представление информации о логистических системах с помощью интеллект-карт.
- Разработка календарных графиков логистических проектов и создание диаграмм Ганта с использованием библиотеки Mermaid и ГИИ.
- Визуализация объектов цепей поставок и связей между ними с помощью ER-диаграмм, а также их автоматизированная разработка с помощью ГИИ.
- Описание динамики логистических процессов в форме UML диаграмм последовательности и диаграмм деятельности, а также их автоматизированное создание с использованием ГИИ.
- Использование диаграмм прецедентов для визуализации логистических процессов, а также декомпозиции цепей поставок.

В России доступно несколько вариантов доступа к системам генеративного искусственного интеллекта, которые можно использовать для решения задач:

- DeepSeek — мощная языковая модель, которая может генерировать код и помогать в решении различных задач (DeepSeek 2025).
- YandexGPT — генеративный текстовый ИИ от Яндекса, который умеет приходить к ответу через цепочку рассуждений, понимает длинный контекст и позволяет вызывать функции для получения данных из внешних источников (Yandex 2025).
- GigaChat — нейросетевая модель от Сбербанка, которая запоминает контекст в многошаговых диалогах, связывает новые вопросы с предыдущей информацией и показывает хорошие результаты в русскоязычных диалогах (СберБанк 2025).

Все эти модели искусственного интеллекта можно обобщённо называть системы генеративного искусственного интеллекта (ГИИ). Они могут самостоятельно выбрать наиболее удобный и доступный им вариант модели.

Диаграммы вариантов использования в UML

На начальных этапах для описания базового функционала программного продукта логистической системы используется диаграмма вариантов использования (диаграмма прецедентов, use case diagram). Она представляет собой элемент нотации UML, который визуальное описывает функциональные требования к системе с точки зрения её взаимодействия с внешними пользователями (актерами) (Галиаскаров Э.Г. 2024).

Основные компоненты диаграммы вариантов использования:

- Актеры — внешние сущности (люди, системы), которые взаимодействуют с системой. Актеры отображаются в виде значков в форме человечков или прямоугольников.
- Прецеденты — функциональные сценарии или действия, которые может выполнять система. Они изображаются в виде овальных фигур.
- Связи — линии, обозначающие взаимодействие между актерами и прецедентами. Используются различные типы связей, включая ассоциации, расширения и обобщения.

Диаграммы вариантов использования помогают разработчикам и другим заинтересованным сторонам понять, какие функции должна выполнять система и как пользователи будут с ней взаимодействовать. Они являются эффективным инструментом на этапе сбора требований при проектировании информационной системы. На диаграмме вариантов использования актеры представлены в виде схематических фигур людей, а прецеденты — в виде эллипсов. Стрелки между элементами показывают направление взаимодействия.

Диаграмма вариантов использования помогает понять, какие возможности предоставляет информационная система и кто может их использовать. Они используются на этапе анализа требований к системе, так как позволяют выявить и графически изобразить основные функциональные возможности информационной системы и потребности пользователей.

В диаграммах вариантов использования поддерживаются различные типы связей между прецедентами и актерами:

ассоциация показывает, что актер использует прецедент, она изображается в виде линии между актёром и прецедентом без стрелки;

включение указывает на то, что один прецедент является частью другого, оно изображается в виде пунктирной линии со стрелкой, указывающей на включаемый прецедент;

расширение означает, что прецедент может быть расширен дополнительными действиями, оно отображается в виде пунктирной линии со стрелкой, направленной от расширяемого прецедента к расширяющему.

Создания диаграммы вариантов использования для двойника логистической системы

Рассмотрим процедуру создания диаграммы вариантов использования на примере проектирования цифрового двойника для логистической компании. Ниже представлена постановка задачи, запросы в систему ГИИ и полученные от нее ответы, а также нарисованный с использованием редактора результат.

Постановка задачи

Логистическая компания ООО «ЛогЛаб» занимается транспортным обслуживанием клиентов в своем регионе. Перевозки выполняются для конкретных грузоотправителей с использованием транспортных средств и водителей компании. Все выполненные перевозки фиксируются в путевых листах. Для улучшения управления и анализа информации о перевозках требуется создание информационной системы. В рамках этой системы предполагается разработка цифрового двойника для ООО «ЛогЛаб». Этот цифровой двойник должен выполнять несколько важных функций:

Представлять справочник марок транспортных средств, которые используются компанией. В этом справочнике будут указаны следующие характеристики каждой марки: (название марки, год выпуска, максимальная грузоподъемность, объем кузова, тип двигателя, тип кузова, пробег)

Справочник транспортных средств, доступных компании. В этом справочнике будут указаны следующие характеристики каждого транспортного средства: (номер автомобиля, марка, год выпуска, состояние, пробег, дата последнего технического обслуживания, идентификатор водителя)

Справочник водителей, работающих на предприятии в настоящее время. По каждому водителю приводится следующая информация: (имя, фамилия, отчество, номер водительского удостоверения, дата выдачи водительского удостоверения, категория водительского удостоверения, дата последней проверки навыков вождения)

Справочник грузоотправителей, обслуживаемых логистической компанией. По каждому грузоотправителю приводится следующая информация: (название компании, адрес, контактный телефон, email, сайт, руководитель, дата заключения договора)

Справочник видов груза, перевозка которых осуществляется компанией. Для каждого вида груза сохраняется следующая информация: (наименование груза, категория груза, вес, объем, размеры, условия хранения, особые требования)

Учет транспортной работы осуществляется с использованием путевых листов. Поэтому создается отдельная таблица для их хранения, в которой для каждого путевого листа сохраняются следующие данные: (номер путевого листа, дата, номер автомобиля, идентификатор водителя, название грузоотправителя, вид груза, расстояние, пройденное по маршруту)

Запрос в ГИИ

Разработай для данной задачи диаграмму вариантов использования для заказчика перевозки в формате PlantUML

Ответ ГИИ

Вот пример диаграммы вариантов использования в формате PlantUML для грузоотправителя в контексте задачи логистической компании ООО «ЛогЛаб»:

```
plantuml
```

```
Copy
```

```
@startuml
```

```
left to right direction
```

```
actor Грузоотправитель
```

```
rectangle "Информационная система ООО «ЛогЛаб»" {
```

```
    Грузоотправитель -- (Просмотр справочника марок ТС)
```

```
    Грузоотправитель -- (Просмотр справочника ТС)
```

```
    Грузоотправитель -- (Просмотр справочника водителей)
```

```
    Грузоотправитель -- (Просмотр справочника грузоотправителей)
```

```
    Грузоотправитель -- (Просмотр справочника видов груза)
```

```
    Грузоотправитель -- (Просмотр путевых листов)
```

```
(Просмотр справочника марок ТС) --> (Справочник марок ТС) : Включает
```

```
(Просмотр справочника ТС) --> (Справочник ТС) : Включает
```

```
(Просмотр справочника водителей) --> (Справочник водителей) : Включает
```

(Просмотр справочника грузоотправителей) --> (Справочник грузоотправителей) : Включает
 (Просмотр справочника видов груза) --> (Справочник видов груза) : Включает
 (Просмотр путевых листов) --> (Таблица путевых листов) : Включает

note right of (Справочник марок ТС)
 Название марки, год выпуска,
 грузоподъемность, объем кузова,
 тип двигателя, тип кузова, пробег
end note

note right of (Справочник ТС)
 Номер автомобиля, марка, год выпуска,
 состояние, пробег, дата ТО,
 идентификатор водителя
end note

note right of (Справочник водителей)
 Имя, фамилия, отчество,
 номер водительского удостоверения,
 дата выдачи, категория,
 дата проверки
end note

note right of (Справочник грузоотправителей)
 Название компании, адрес, телефон,
 email, сайт, руководитель,
 дата договора
end note

note right of (Справочник видов груза)
 Наименование груза, категория,
 вес, объем, размеры,
 условия хранения, особые требования
end note

note right of (Таблица путевых листов)
 Номер путевого листа, дата,
 номер автомобиля, идентификатор водителя,
 грузоподрядчик, вид груза, расстояние
end note

}

@enduml

Описание диаграммы:

1. Грузоотправитель — основной актор, который взаимодействует с системой.
2. Просмотр справочников и таблиц — варианты использования, которые доступны грузоотправителю.
3. Справочники и таблицы — содержат информацию о марках ТС, транспортных средствах, водителях, грузоотправителях, видах груза и путевых листах.
4. Примечания (notes) — описывают атрибуты каждого справочника или таблицы.

Эта диаграмма наглядно показывает, как грузоотправитель взаимодействует с системой и какие данные он может просматривать.

Описание полученного результата

Полученный цифровой двойник логистической компании ООО «ЛогЛаб» предлагает различные варианты использования информационной системы с позиций грузоотправителей. С использованием этого цифрового двойника грузоотправители смогут фиксировать в системе создание заказов на перевозку и другие ее параметры. Система процесс позволяет грузоотправителям сохранить все необходимые детали заказа.

Вторым вариантом использования является отслеживание текущего состояния перевозки. Грузоотправители могут отслеживать перемещение своих грузов в режиме реального времени, что обеспечивает прозрачность цепи поставок и непрерывный контроль над процессом. Грузоотправители смогут своевременно реагировать на любые изменения и оперативно решать возникающие вопросы.

Также цифровой двойник предоставляет возможность управления информацией о грузе. Это будет востребовано для сложных заказов, в которых требуется постоянное обновление информации о текущем состоянии груза. Таким образом, цифровой двойник обеспечивает грузоотправителям удобство и контроль над различными аспектами процесса перевозки.

На рисунке 1 приведен результат построения диаграммы вариантов использования для полученной из системы генеративного искусственного интеллекта описания диаграммы.



Рис.1. Результат построения диаграммы вариантов использования цифрового двойника логистической системы

В результате была получена диаграмма вариантов использования для создания информационной системы цифрового двойника логистической системы. Несмотря на то, что предложенный вариант не идеален, но он может служить хорошей отправной точкой для дальнейшей работы системному аналитику. Уже сейчас ГИИ показывает хорошие результаты для таких задач, и в будущем он будет напрямую интегрирован непосредственно в программные средства для создания информационных систем.

Заключение

1. Широкое использование искусственного интеллекта и цифровых технологий в логистике формирует основу концепции «Логистика 4.0», которая аналогична принципам «Индустрии 4.0». Эта концепция направлена на цифровизацию логистических процессов, повышение их эффективности и устойчивости цепочек поставок, что особенно актуально в условиях глобальных вызовов.

2. Внедрение таких технологий, как интернет вещей (IoT), киберфизические системы, большие данные, облачные вычисления и искусственный интеллект, позволяет создавать цифровые двойники логистических систем. Это обеспечивает прозрачность, безопасность и оперативность управления цепочками поставок, а также способствует повышению экономической, социальной и экологической устойчивости.

3. Создание цифровых двойников, связанных с реальными логистическими системами, позволяет моделировать и оптимизировать процессы в режиме реального времени. Широкое использование устройств отслеживания, включая IoT-устройства, обеспечивает контроль над перемещением грузов на всех этапах, что повышает эффективность и снижает издержки.

4. «Логистика 4.0» фокусируется на удовлетворении индивидуальных потребностей клиентов без увеличения затрат, что достигается за счет цифровизации и использования интеллектуальных технологий. Такой подход позволяет адаптировать логистические процессы под изменяющиеся требования рынка, обеспечивая конкурентоспособность и устойчивое развитие компаний.

5. UML активно используется для моделирования логистических систем различного типа, включая пассажирский транспорт, транспортную логистику и планирование производства. Российские и зарубежные исследования демонстрируют его универсальность и применимость для описания архитектуры, функциональности и поведения сложных логистических систем.

6. В логистических исследованиях применяются различные типы диаграмм UML, такие как диаграммы вариантов использования, классов, последовательностей, компонентов и действий. Эти диаграммы позволяют визуализировать ключевые аспекты логистической системы, включая взаимодействие объектов, процессы принятия решений и структуру данных, что упрощает проектирование информационных систем поддержки.

7. UML успешно интегрируется с другими подходами, например с такими как модель спецификации дискретных событий (DEVS), что расширяет возможности для концептуального моделирования логистических систем. Такая интеграция позволяет улучшить точность и детализацию моделей, а также автоматизировать процессы тестирования и выявления уязвимостей.

8. В статье рассматривается пример применения генеративного искусственного интеллекта (ГИИ) для создания диаграммы прецедентов информационной системы цифрового двойника логистической системы. Хотя предложенный вариант не является идеальным, но он может послужить отправной точкой для дальнейшей разработки информационной системы. ГИИ демонстрирует хорошие результаты в решении подобных задач и в будущем будет интегрирован непосредственно в программные средства для создания информационных систем.

Литература

1. Абд Алхади, Х. 2023. «Моделирование информационной системы транспортной логистики / Х. Абд Алхади, Ю. В. Минин // Информационные процессы, системы и технологии. – 2023. – Т. 4, № 1(25). – С. 22-28. – DOI 10.52529/27821617_2023_4_1_22. – EDN OBSVLL.»
2. Галиаскаров Э.Г. 2024. Анализ и проектирование систем с использованием UML : учебное пособие для вузов / Э. Г. Галиаскаров, А. С. Воробьев.— Москва: Издательство Юрайт, 2024.— 125с.
3. Иващенко, Н. Ю. 2006. «Применение унифицированного языка моделирования для проектирования логистических систем пассажирского транспорта / Н. Ю. Иващенко, В. Н. Трегубов, Е. В. Феклин // Вестник Саратовского государственного технического университета. – 2006. – Т. 4, № 2(17). – С. 107-112. – EDN ZVKHYH.»
4. СберБанк. 2025. «GigaChat — русскоязычная нейросеть от Сбера». giga.chat. <https://giga.chat/> (7 январь 2025 г.).
5. Трегубов, В. 2024. «Перспективные направления исследований использования генеративного искусственного интеллекта в маркетинге / В. Н. Трегубов // International Journal of Open Information Technologies. – 2024. – Т. 12, № 5. – С. 23-32. – EDN RNREUK.»
6. Amr, Mohamed, Mohamed Ezzat, и Sally Kassem. 2019. «Logistics 4.0: Definition and Historical Background». В IEEE, 46–49. doi:10.1109/niels.2019.8909314.
7. Dakic, Dusanka, Darko Stefanovic, Teodora Lolic, Srdjan Sladojevic, и Andras Anderla. 2018. «Production planning business process modelling using UML class diagram». В IEEE, 1–6. doi:10.1109/in-foteh.2018.8345535.
8. DeepSeek. 2025. «DeepSeek». <https://chat.deepseek.com> (7 январь 2025 г.).
9. Evtodieva, T. E., D. V. Chernova, N. V. Ivanova, и N. S. Kisteneva. 2019. «Logistics 4.0». Contributions to Economics: 207–19. doi:10.1007/978-3-030-11754-2_16.
10. Gupta, Himanshu, Avinash Kumar Yadav, Simonov Kusi-Sarpong, Sharfuddin Ahmed Khan, и Shashi Chandra Sharma. 2022. «Strategies to overcome barriers to innovative digitalisation technologies for supply chain logistics resilience during pandemic». Technology in Society 69: 101970. doi:10.1016/j.techsoc.2022.101970.
11. Hrouga, Mustapha, и Abdelkader Sbihi. 2023. «Logistics 4.0 for Supply Chain Performance: Perspectives from a Retailing Case Study». Business Process Management Journal 29(6): 1892–1919. doi:10.1108/BPMJ-03-2023-0183.
12. Khan, Shahbaz, Rubee Singh, José Carlos Sá, Gilberto Santos, и Luís Pinto Ferreira. 2022. «Modeling of Determinants of Logistics 4.0 Adoption: Insights from Developing Countries». Machines 10(12): 1242. doi:10.3390/machines10121242.
13. Le, Tho V., и Ruoling Fan. 2024. «Digital Twins for Logistics and Supply Chain Systems: Literature Review, Conceptual Framework, Research Potential, and Practical Challenges». Computers & Industrial Engineering 187: 109768. doi:10.1016/j.cie.2023.109768.
14. Liu, Yu, Shenle Pan, и Eric Ballot. 2024. «Unveiling the Potential of Digital Twins in Logistics and Supply Chain Management: Services, Capabilities, and Research Opportunities». Digital Engineering 3: 100025. doi:10.1016/j.dte.2024.100025.
15. PlantText. 2025. «PlantText UML Editor». <https://www.planttext.com/> (7 январь 2025 г.).
16. PlantUML. 2025. «Инструмент с открытым исходным кодом, использующий простые текстовые описания для рисования UML-диаграмм.» <https://plantuml.com/ru/> (7 январь 2025 г.).
17. Szymańska. 2017. «Logistics 4.0 - a new paradigm or set of known solutions?» doi:10.21008/J.2083-4950.2017.7.4.2.
18. Winkelhaus, Sven, и Eric H. Grosse. 2019. «Logistics 4.0: a systematic review towards a new logistics system». International Journal of Production Research 58(1): 18–43. doi:10.1080/00207543.2019.1612964.
19. Yandex. 2025. «YandexGPT 4». <https://ya.ru/ai/gpt-4> (7 январь 2025 г.).
20. Yonghua Zhou, Yuliu Chen, и Huapu Lu. 2004. «UML-based systems integration modeling technique for the design and development of intelligent transportation management system». В 2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583), The Hague, Netherlands: IEEE, 6061–66. doi:10.1109/ICSMC.2004.1401349.
21. Zhiyong, Zhang, Liu Jie, и Zhang Xinhui. 2013. «UML Modeling for Dynamic Logistics System Based on DEVS». TELKOMNIKA Indonesian Journal of Electrical Engineering 11(4). doi:10.11591/telkomnika.v11i4.2397.

References in Cyrillics

1. Abd Alhadi, X. 2023. «Modelirovanie informacionnoj sistemy transportnoj logistiki / H. Abd Alhadi, Yu. V. Minin // Informacionnye processy, sistemy i tekhnologii. – 2023. – Т. 4, № 1(25). – S. 22-28. – DOI 10.52529/27821617_2023_4_1_22. – EDN OBSVLL.»
2. Galiaskarov E.G. 2024. Analiz i proektirovanie sistem s ispol'zovaniem UML : uchebnoe posobie dlya vuzov / E. G. Galiaskarov, A. S. Vorob'ev.— Moskva: Izdatel'stvo Yurajt, 2024.— 125s.
3. Ivashchenko, N. Yu. 2006. «Primenenie unificirovannogo yazyka modelirovaniya dlya proektirovaniya logisticheskikh sistem passazhirskogo transporta / N. Yu. Ivashchenko, V. N. Tregubov, E. V. Feklin // Vestnik Saratovskogo gosudarstvennogo tekhnicheskogo universiteta. – 2006. – Т. 4, № 2(17). – S. 107-112. – EDN ZVKHYH.»
4. SberBank. 2025. «GigaChat — russkoyazychnaya nejroset' ot Sbera». giga.chat. <https://giga.chat/> (7 yanvar' 2025 g.).
5. Tregubov, V. 2024. «Perspektivnye napravleniya issledovaniy ispol'zovaniya generativnogo iskusstvennogo intellekta v marketinge / V. N. Tregubov // International Journal of Open Information Technologies. – 2024. – Т. 12, № 5. – S. 23-32. – EDN RNREUK.»

Ключевые слова

Логистика 4.0, генеративный искусственный интеллект, ГИИ, UML, цифровой двойник

Трегубов В.Н., д.э.н.,

профессор кафедры производственного менеджмента, ФГБОУ ВО Саратовский государственный технический университет имени Гагарина Ю.А., г. Саратов, Российская Федерация, e-mail: tregubovvn@outlook.com

SPIN-код: 5528-8060 ORCID: 0000-0002-7793-3605

Матушкин М.А., д.э.н.,

профессор кафедры производственного менеджмента, ФГБОУ ВО Саратовский государственный технический университет имени Гагарина Ю.А., г. Саратов, Российская Федерация, e-mail: matushkinma@mail.ru

SPIN-код: 5056-6300 ORCID: 0000-0001-6803-1178

Tregubov V.N., Matushkin M.A., The use of generative artificial intelligence technology for the design of digital twins in logistics

Keywords

Logistics 4.0, generative artificial intelligence, AI, UML, digital twins.

DOI: 10.34706/DE-2025-01-01

JEL classification C8 Методология сбора и оценки данных; компьютерные программы; O33 – Научно-технический прогресс: этапы и последствия; процесс распространения

Abstract

The authors explore the concept of "Logistics 4.0", which is formed on the principles of "Industry 4.0" and aims at digital transformation of logistics processes using artificial intelligence and digital technologies. The use of Internet of Things (IoT), cyber-physical systems, big data and cloud computing in logistics makes it possible to create digital twins of logistics systems. This contributes to economic, social and environmental sustainability by providing transparency, security and agility in supply chain management. An effective modelling tool for creating digital twins is the Unified Modelling Language (UML). Russian and foreign studies demonstrate its good applicability for describing the architecture, functionality and behaviour of logistics systems. Integration of UML with other methods, such as the Discrete Event Specification (DEVS) model, expands its capabilities as a conceptual modelling tool. This article discusses the application of generative artificial intelligence (AI) to create precedence diagrams for the development of a digital twin of a logistics system. Although the proposed GIA solution is not perfect, it demonstrates the potential of GIA in solving such problems, and the created model can serve as a starting point for further information system development. In the future, the integration of GII into software tools for creating information systems will open up new possibilities for shaping logistics processes.

УДК: 004

1.2. Практическая реализация методологии CODE

Зекирьяев Р.Т., Болбаков Р.Г.,
МИРЭА – Российский технологический университет, Москва, Россия

В статье рассматривается интеграция методологии CODE (Codebase Operations and Development Practices) в процесс разработки программного обеспечения через создание плагина для интегрированной среды разработки IntelliJ IDEA. Методология CODE, основанная на адаптации цикла Деминга-Шухарта (PDCA), включает четыре этапа: Prepare, Develop, Control и Apply, каждый из которых направлен на стандартизацию и автоматизацию управления кодовой базой.

В работе обоснована необходимость практической реализации методологии в виде инструмента, интегрированного в среду разработки. Показано, что создание плагина для среды разработки позволяет разработчикам эффективно соблюдать стандарты, улучшать качество кода и ускорять выполнение повседневных задач. В статье описаны логика работы плагина, ключевые функции на каждом этапе методологии и примеры действий, иницируемых пользователем, таких как создание новых веток, проверка соблюдения стиля написания кода, запуск тестов и управление процессом слияния изменений.

Особое внимание уделено начальной стадии разработки плагина, включая настройки проекта и использование API IntelliJ IDEA для взаимодействия с системами контроля версий, анализа кода и тестирования. Предложены концептуальные примеры кода, иллюстрирующие реализацию функциональности.

Рассмотренные аспекты демонстрируют потенциал использования плагина для упрощения внедрения методологии CODE в реальную практику командной разработки. Продолжением этого исследования может стать непосредственная техническая реализация плагина. Представленные идеи закладывают основу для практического применения методологии CODE, направленного на повышение эффективности и стандартизации процессов разработки.

Введение

В научной статье, посвящённой теоретическому описанию методологии CODE (Codebase Operations and Development Practices) [1], был введён систематизированный подход к работе с кодовой базой, объединивший разрозненные практики разработки, проверки и интеграции изменений. Эта методология была предложена как решение проблемы отсутствия единого стандарта, охватывающего полный цикл работы с исходным кодом в профессиональной среде. Описанный в работе цикл Prepare-Develop-Control-Apply, вдохновлённый принципами Деминга-Шухарта [2], уже демонстрирует свою концептуальную состоятельность и предоставляет разработчикам чёткие рекомендации на каждом этапе.

Теоретическое описание, каким бы детальным оно ни было, не даёт возможности немедленного внедрения и апробации технологии. Современные требования разработки требуют практических инструментов, которые способны автоматически контролировать выполнение этапов, адаптироваться к разным условиям проектов и минимизировать человеческий фактор. Именно поэтому возникает необходимость создания инструментальной реализации методологии CODE, позволяющей сделать её применение удобным, интуитивно понятным и масштабируемым.

Работа с кодовой базой программного продукта связана с повседневной рутинной: создание веток, написание кода, соблюдение стиля написания кода (далее код-стайла) [3], тестирование и проверка написанного инкремента (далее код-ревью) [4]. Каждое из этих действий может быть стандартизировано, но для этого разработчику требуется вспомогательный инструмент, который будет направлять его, отслеживать этапы и предоставлять рекомендации.

Проблемы, связанные с применением методологии, как правило, обусловлены субъективным подходом команд к её внедрению. Например, разработчик может забыть отследить покрытие тестами только что разработанного инкремента или не проверить соответствие кода установленным стандартам. Такие ошибки могут становиться систематическими, если отсутствует надлежащий контроль. Таким образом, необходимость автоматизации этапов методологии становится очевидной.

Создание модуля, или, рассуждая в контексте расширений для сред разработки, плагина для интегрированной среды разработки (IDE) [5], позволит не только облегчить процесс внедрения CODE, но и значительно ускорить адаптацию новых участников команды. Инструментальная реализация сделает методологию не просто сводом правил, а функциональной частью рабочего процесса, которую можно использовать

ежедневно. В то же время она сохраняет гибкость и адаптивность, позволяя командам вносить изменения и адаптировать подход под конкретные задачи.

Практическая реализация методологии также играет важную роль в обеспечении её жизнеспособности. Если CODE может быть применена в различных сценариях — от небольших стартапов до крупных корпоративных проектов — она имеет шансы стать стандартом в индустрии. Инструментальная поддержка обеспечит доказательство её эффективности, демонстрируя реальные результаты, такие как сокращение времени на доработки, повышение качества кода и улучшение коммуникации внутри команды.

Таким образом, необходимость практической реализации методологии CODE становится очевидной. Это шаг, который переводит теоретическую концепцию в реальный рабочий процесс, облегчая внедрение и гарантируя соответствие заявленным принципам. А наилучший способ внедрить полезную практику в существующий процесс — максимально автоматизировать применение этой практики. Основным инструментом разработчика программного обеспечения — это IDE или среда разработки [5], поэтому наибольшей эффективностью применения методологии и охвата процессов создания программного продукта можно добиться модернизацией IDE. Основу расширений для среды разработки составляют плагины [6]. Поэтому далее будет рассмотрен выбор среды разработки для реализации инструмента и рассмотрен процесс интеграции.

Интеграция методологии в среду разработки

Для эффективного внедрения методологии CODE в повседневную практику разработчиков необходим инструмент, который будет интегрирован непосредственно в их рабочую среду. Такой подход обеспечит непрерывность процессов и минимизирует дополнительные усилия, связанные с переключением между различными приложениями или инструментами. В этом контексте интеграция методологии в виде модуля для среды разработки представляется наиболее целесообразным решением.

Интеграция методологии CODE в виде модуля станет важным шагом, который позволит разработчикам соблюдать установленные правила управления кодовой базой непосредственно в ходе своей повседневной работы. В процессе разработки программного обеспечения, где задачи часто связаны с созданием и поддержкой сложной структуры кода, стандартизированные подходы позволяют минимизировать количество ошибок, ускорить адаптацию новых участников команды и обеспечить соблюдение стандартов качества. Такой инструмент, как плагин, становится связующим звеном между теоретической моделью методологии и её практическим применением, делая её неотъемлемой частью рабочего процесса. Плагин не просто напоминает разработчикам о правилах, но активно направляет их на каждом этапе выполнения методологии, помогая им избегать распространённых ошибок, поддерживать стандарты и следовать лучшим практикам.

Среди множества существующих сред разработки IntelliJ IDEA выделяется как удобная платформа для реализации подобного инструмента. Её универсальность, мощные встроенные функции и возможность гибкого расширения делают её подходящей для интеграции методологии CODE. IntelliJ IDEA предоставляет разработчикам интеллектуальную поддержку кода, которая включает в себя автодополнение, статический анализ, подсказки по рефакторингу и автоматизацию рутинных задач [7]. Эти функции позволяют минимизировать затраты времени на исправление ошибок, обеспечивая высокое качество разрабатываемого программного обеспечения. Кроме того, встроенные средства работы с системами контроля версий, такими как Git, позволяют автоматизировать процессы управления изменениями в кодовой базе, что идеально сочетается с требованиями методологии CODE.

Ещё одним преимуществом IntelliJ IDEA является её архитектура, ориентированная на создание и интеграцию плагинов. Используя предоставленный компанией JetBrains SDK [8], разработчики могут создавать плагины, которые бесшовно интегрируются в интерфейс и функциональность среды. Это означает, что плагин для реализации CODE будет не просто дополнительным инструментом, а органичной частью рабочей среды разработчика. Интерфейс IntelliJ IDEA позволяет пользователям быстро освоить работу с новым функционалом, не теряя времени на сложные настройки или обучение.

Ценность IntelliJ IDEA представляет и для команд, работающих с различными языками программирования. Её широкая поддержка технологий делает её универсальным инструментом, способным удовлетворить потребности разработчиков вне зависимости от того, используют ли они Java, Kotlin, Python, JavaScript или другой язык. Это особенно важно в контексте современной разработки, где проекты часто требуют междисциплинарного подхода и взаимодействия между командами, использующими разные технологии. Универсальность IntelliJ IDEA позволяет интегрировать методологию CODE в проекты любого масштаба, от небольших стартапов до крупных корпоративных решений.

Интеграция методологии CODE в среду разработки через плагин для IntelliJ IDEA открывает значительные перспективы для оптимизации процессов разработки и управления кодовой базой. Это решение позволит командам стандартизировать свои подходы, минимизировать негативное влияние человеческого фактора и ускорить разработку проектов. Кроме того, реализованный плагин может быть адаптирован под специфические нужды различных проектов, что сделает его универсальным и перспективным инструментом для профессиональной среды.

Перед тем, как перейти к описанию логики работы плагина, вспомним основные концепции и идеи методологии CODE.

Методология CODE

Методология наследует структуру цикла Деминга-Шухарта PDCA (Plan-Do-Check-Act). В рамках методологии CODE цикл PDCA имеет структуру: Prepare-Develop-Control-Apply. На первом этапе выполняется

подготовка к разработке, подготовка включает в себя информацию, необходимую уже непосредственно при работе с кодом. На втором этапе методологии выполняется разработка по характерным для конкретного проекта правилам. Третий этап – проверка, он включает в себя принятые практики анализа написанного кода. Четвёртый этап включает в себя применение созданных изменений. Сам цикл представлен на рисунке 1. Рассмотрим, что конкретно включает в себя каждый из этапов.

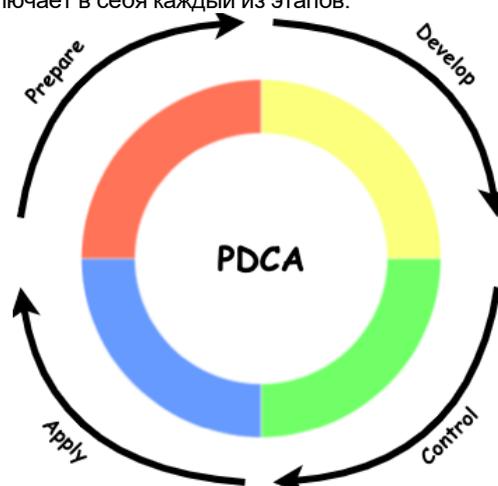


Рисунок 1 – Цикл Prepare-Develop-Control-Apply

Prepare (Подготовка)

При работе с большими объёмами кода используются системы контроля версий. Для эффективной работы с системой в параллели с одной кодовой базой используются различные модели ветвления (Branching) – Feature Based Development (FBD), Trunk Based Development (TBD). Очень важно, чтобы все разработчики понимали, какая модель используется в их кодовой базе, как выполнять доработки и слияние своих доработок с общей кодовой базой, сколько коммитов должно быть в локальной ветке. Поэтому самым первым шагом является изучение модели ветвления и всех принятых практик по работе с системой контроля версий – отведение веток для релиза, хотфиксов [9] и простых доработок. Основа этого этапа – описанные практики работы с системой контроля версий.

Develop (Разработка)

После выполнения подготовки, создания собственного пространства для работы в системе контроля версий, наступает черёд непосредственно разработки. Разработка также не должна носить хаотичный характер – поэтому при написании кода хорошей практикой является следование стилю написания кода, или Code style (CS). Code style содержит в себе информацию об именовании сущностей в проекте, расположении свойств и методов в рамках класса, рекомендации по оформлению кода и написанию комментариев. В рамках Code style также могут быть даны рекомендации по написанию методов или реализации задач, специфичных для конкретной кодовой базы. Здесь же могут описываться используемые в разработке шаблоны проектирования. Ключевой регламент этапа разработки – стиль написания кода.

Control (Проверка)

За разработкой следует этап проверки, или тестирования. На этом этапе код уже приобрёл бизнес-ценность [10] и её следует проверить на соответствие ожиданиям. В первую очередь выполняется ручная проверка разработанной функциональности. Далее на этом же этапе выполняется автоматизация тестирования через написание тестов. Например, хорошей практикой будет следование упомянутой ранее пирамиде тестирования – написание малой доли End to end (E2E) тестов, чуть большего числа интеграционных тестов и большого числа, относительно предыдущих двух типов, Unit-тестов. На этом этапе следует описать практики автоматизированного тестирования.

Завершающий этап цикла – применение разработанных изменений. В соответствии с практиками работы с системой контроля версий создаётся запрос на слияние с основной кодовой базой созданных изменений (Pull/Merge Request – PR/MR). Создание таких запросов также желательно описывать, чтобы избежать различий в форматах, например: Название, описание, пояснения к доработкам, размер изменений в одном запросе. За этапом создания запроса следует Code Review (CR) – проверка вносимых изменений другими разработчиками проекта. CR может быть очень сложным процессом, поэтому следует заранее договориться о правилах его проведения – просматривать поступившие запросы на CR за конкретный, ограниченный во времени срок, абстрагироваться от эмоционального контекста, если он мешает качественному просмотру доработок [11].

Это ключевые концепции каждого этапа методологии CODE. Далее рассмотрим возможную логику работы плагина, включая действия пользователя, инициирующие переходы между этапами методологии CODE, и соответствующие реакции системы.

Логика работы плагина

Методология CODE может использоваться в реальных проектах уже в текущем виде, в том, в котором она описана в первой научной статье цикла, и в том же виде, что она описана выше. В таком варианте исполнения она может находиться в корневом пакете проекта и выполнять роль документации для введения в контекст проекта (онбординга) [12] новых разработчиков или для обращения уже работающих участников проекта с целью вспомнить основные договорённости, зафиксированные для работы с кодовой базой. Но если автоматизировать пункты, описанные в методологии, то можно получить мощный инструмент для регулярной поддержки качества и управления кодовой базой. Перейдём к описанию возможных вариантов реализации методологии CODE как плагина для среды разработки.

Плагин, реализующий методологию CODE в среде IntelliJ IDEA, направлен на упрощение и автоматизацию процессов разработки, он обеспечивает последовательное выполнение этапов методологии: Prepare, Develop, Control и Apply.

Этап Prepare открывает цикл создания инкремента кодовой базы, на нём закладывается основа для эффективной работы с кодом. На этом этапе плагин может предоставлять разработчику возможность создать новую ветку для работы над функциональностью или исправлением ошибок. Плагин может отображать диалоговое окно с предложением выбрать тип ветки, например, feature, bugfix или hotfix, основываясь на принятой модели ветвления проекта [13]. После выбора типа ветки пользователь вводит её имя. Здесь плагин может играть роль контролирующего механизма: он проверяет, соответствует ли имя ветки заданным стандартам. Например, если имя ветки не начинается с префикса feature/ или содержит запрещённые символы, плагин может уведомлять разработчика об ошибке и предлагать её исправить. В дополнение к этому плагин может предлагать ссылку на внутреннюю документацию проекта, где описаны правила именования веток, или открывать файл в проекте, в котором подробно изложены эти стандарты. Это может позволить разработчику сразу ознакомиться с необходимыми требованиями без необходимости покидать среду разработки.

На этапе Develop плагин обеспечивает поддержку соблюдения стандартов кодирования и стиля. Это может достигаться благодаря интеграции с инструментами статического анализа кода [14]. После каждого сохранения файла плагин может инициировать проверку текущего состояния кода, выявляя потенциальные нарушения стиля, использование устаревших конструкций или наличие логических ошибок. Если такие нарушения обнаруживаются, плагин может уведомлять пользователя через всплывающее окно или вывод сообщений в консоли IDE. При этом плагин может не только указывать на проблему, но и предлагать варианты её решения. Например, если код не соответствует принятому стилю, плагин может предоставлять кнопку для автоматического форматирования, используя заранее настроенные правила. Для более сложных случаев, таких как обнаружение устаревших методов, плагин, используя средства IDE, может предлагать ссылки на документацию или ресурсы, объясняющие, почему метод больше не рекомендуется и какие альтернативы существуют. Таким образом, плагин может не только исправлять ошибки, но и способствовать обучению разработчика, предоставляя ему доступ к полезной информации.

Этап Control представляет собой центральный элемент процесса проверки качества кода. После завершения разработки функциональности плагин может инициировать процесс тестирования. Он может автоматически запускать все модульные тесты, настроенные в проекте, и проверять уровень покрытия кода тестами. Если тесты успешно проходят, плагин может уведомлять разработчика и предлагать следующий шаг — создание PR для последующего код-ревью. Здесь плагин может предоставлять шаблон для описания изменений, в который включаются автоматически сгенерированные метрики, такие как количество новых строк кода, уровень покрытия тестами и список затронутых файлов. Это поможет разработчику представить свои изменения в структурированном виде. Если же тесты не пройдены, плагин может формировать подробный отчёт, включающий описание ошибок, их местоположение и возможные причины. Отчёт может быть представлен в виде HTML-документа с гиперссылками, ведущими к строкам кода, вызвавшим ошибки. Кроме того, плагин может предоставлять рекомендации по устранению проблем, основанные на анализе тестов и стандартных шаблонах решений.

На завершающем этапе Apply плагин играет важную роль в интеграции изменений в основную кодовую базу. После успешного прохождения CR плагин может проверять, были ли исправлены все замечания, оставленные ревьюерами, и предлагать разработчику завершить процесс слияния. Перед слиянием плагин может автоматически обновлять ветку разработки последними изменениями из основной ветки, чтобы минимизировать вероятность конфликтов. Если конфликты всё же возникают, плагин может предоставлять инструменты для их разрешения, такие как визуальный редактор, позволяющий наглядно сравнивать конфликтующие версии. После успешного завершения слияния плагин может предложить удалить временную ветку, чтобы сохранить порядок в репозитории. Это действие может сопровождаться напоминанием о необходимости обновления документации, связанной с реализованной функциональностью, если такое требуется. В качестве дополнительной опции плагин может отправлять уведомления другим участникам команды о завершении работы над задачей, предоставляя ссылку на итоговый PR и описание внесённых изменений.

Для каждого из описанных этапов плагин может работать в двух режимах: индикативном и активном. В индикативном режиме плагин может предоставлять пользователю подсказки и ссылки на ресурсы, такие как методология или правила проекта. Например, на этапе подготовки он может открывать файл CODE.md или раздел документации, где подробно описаны правила именования веток. В активном режиме плагин может выполнять проверки, предоставлять рекомендации и инициировать автоматические действия, такие как запуск тестов или форматирование кода. Это может делать его универсальным инструментом, подходящим как

для опытных разработчиков, которым нужна только справочная информация, так и для начинающих, которым требуется активная поддержка на каждом этапе.

Таким образом, плагин может стать не просто инструментом, а полноценным проводником методологии CODE, обеспечивая её интеграцию в повседневную работу разработчиков.

Конфигурация плагина

Разработка плагина для IntelliJ IDEA начинается с создания нового проекта с использованием Gradle [15] и с подключением необходимых зависимостей. Настроим плагин для работы не только с IntelliJ, но и с Android Studio [16], модификацией среды для Android-разработки. Выполним этот шаг для расширения охвата разработчиков [17], которые могут воспользоваться потенциальными возможностями будущего плагина, реализующего методологию CODE.

Приступим непосредственно к конфигурации будущего плагина. Принцип действия плагина описан в файлах конфигурации. После создания пустого проекта плагина в IntelliJ IDEA в файле `build.gradle.kts` указываются основные параметры плагина. Рассмотрим этот файл, он представлен в листинге 1.

Листинг 1 – Часть конфигурации плагина в файле `build.gradle.kts`

```
plugins {
    id("java")
    id("org.jetbrains.kotlin.jvm") version "1.9.23"
    id("org.jetbrains.intellij") version "1.17.3"
}

group = "ru.mirea"
version = "1.0.0"

repositories {
    mavenCentral()
}

intellij {
    version.set("2023.2.6")
    plugins.set(listOf("Kotlin", "java", "android"))
}
```

Версии подключённых плагинов актуальны на момент написания работы, однако могут меняться в будущем.

В данном файле подключаются следующие плагины:

Java-плагин (`id("java")`): обеспечивает поддержку разработки на языке Java, предоставляя необходимые инструменты для компиляции и сборки Java-проектов.

Kotlin JVM плагин (`id("org.jetbrains.kotlin.jvm")`): добавляет поддержку языка Kotlin в проекты, предназначенные для Java Virtual Machine (JVM), что позволяет писать код на Kotlin и компилировать его в байт-код JVM.

IntelliJ плагин (`id("org.jetbrains.intellij")`): предназначен для разработки плагинов под платформу IntelliJ, включая Android Studio. Он упрощает процесс создания, сборки и тестирования плагинов, обеспечивая интеграцию с экосистемой JetBrains.

Параметры `group` и `version` определяют группу и версию создаваемого плагина соответственно, что необходимо для его идентификации и управления версиями.

В разделе `repositories` указывается репозиторий Maven Central [18], из которого будут загружаться необходимые зависимости.

Блок `intellij` настраивает параметры среды разработки, для которой создаётся плагин:

`version.set("2023.2.6")`: определяет версию IntelliJ IDEA, с которой будет совместим плагин.

`plugins.set(listOf("Kotlin", "java", "android"))`: задаёт список плагинов, необходимых для работы разрабатываемого плагина. В данном случае подключаются плагины для поддержки Kotlin, Java и Android, что обеспечивает совместимость плагина как с IntelliJ IDEA, так и с Android Studio.

Особенностью разработки плагина для Android Studio является необходимость учёта специфики Android-разработки, включая работу с Android SDK [19], ресурсами и специфическими инструментами. Поэтому важно убедиться, что в настройках плагина подключены соответствующие модули и зависимости, обеспечивающие корректную работу в среде Android Studio.

Далее создаётся файл манифеста плагина `plugin.xml`, в котором определяются действия, расширения и слушатели, предоставляемые плагином:

```
Листинг 2 - plugin.xml
<idea-plugin>
    <id>ru.mirea.codeplugin</id>
```

```

<name>CODE Methodology Plugin</name>
<version>1.0.0</version>
<description>
  Плагин для интеграции методологии CODE в IntelliJ IDEA.
</description>
<depends>com.intellij.modules.platform</depends>
<depends>com.intellij.java</depends>
<depends>org.jetbrains.kotlin </depends>
<depends>org.jetbrains.android </depends>

<extensions defaultExtensionNs="com.intellij">
  <!-- Определение действий и расширений -->
</extensions>

<actions>
  <!-- Определение пользовательских действий -->
</actions>
</idea-plugin>

```

В этом файле указываются идентификатор плагина, его название, версия, информация о поставщике (это поле удалено из листинга), а также зависимости и расширения, которые плагин добавляет в среду разработки. Особое внимание нужно уделить блокам `<depends>`, т.к. они предоставляют зависимости, с которыми в дальнейшем и предстоит реализовать плагин.

После настройки проекта и определения основных компонентов, таких как файл манифеста `plugin.xml`, следующим шагом будет реализация функциональности, соответствующей каждому этапу методологии: `Prepare`, `Develop`, `Control` и `Apply`. В рамках этой работы будут очень кратко рассмотрены варианты действий для каждого этапа, но без конкретной технической реализации.

Автоматизация этапов методологии CODE основывается на действиях разработчика, которые являются триггерами для перехода между этапами. Плагин будет отслеживать эти действия в процессе работы и предоставлять необходимые рекомендации или выполнять автоматизированные операции. Каждый этап — `Prepare`, `Develop`, `Control`, `Apply` — инициируется конкретными действиями пользователя и сопровождается логической реакцией системы, что позволяет сделать рабочий процесс структурированным и контролируемым.

Этап `Prepare`. На этапе подготовки плагин реагирует на создание новой ветки в системе контроля версий. В большинстве современных команд создание веток — это первый шаг при начале работы над новой задачей. Для реализации этого процесса плагин должен уметь взаимодействовать с системой контроля версий Git [20] через соответствующие API IntelliJ IDEA.

Пользователь инициирует этап `Prepare` либо выбирая пункт из меню плагина «Создать ветку» или вызывая соответствующую команду через горячие клавиши, либо через введение имени новой ветки в окне создания ветки, которое отображается при активации плагина. Также начало выполнения этапа возможно при выполнении команды `fetch` для обновления локального репозитория.

На данном этапе плагин выполняет несколько последовательных операций. Сначала проверяется корректность введённого имени ветки. Например, имя ветки должно соответствовать определённому паттерну, принятому в проекте (например, `feature/`, `bugfix/` или `hotfix/`). Если имя ветки не соответствует стандартам, плагин уведомляет пользователя о необходимости корректировки.

После успешного создания ветки система фиксирует событие и автоматически выполняет `git-checkout` для переключения на новую ветку. Пример реализации этого шага может выглядеть следующим образом:

```

Листинг 3 – Псевдокод варианта реализации этапа Prepare
val project: Project = getProject(...)
val git = Git.getInstance()
val branchName = "feature/new-feature"

// Проверка имени ветки
if (isBranchNameValid(branchName)) {
  git.createBranch(project, branchName)
  git.checkout(project, branchName)
} else {
  notifyUser("Имя ветки не соответствует стандарту")
}

```

Завершение создания ветки является сигналом о переходе на следующий этап методологии.

Этап `Develop`. Этап разработки начинается сразу после успешного создания ветки и выполнения команды `checkout`. На данном этапе действия пользователя сводятся к редактированию исходного кода, что

плагин отслеживает по событиям изменения файлов в редакторе. Каждое сохранение файла является триггером для выполнения дополнительных проверок на соответствие стандартам кодирования и стилю.

Интеграция со статическим анализатором (например, CheckStyle или встроенные инспекции IntelliJ IDEA) позволяет плагину анализировать код и автоматически выявлять нарушения. Результаты анализа отображаются в виде уведомлений в панели задач IDE или встроенном окне проблем. Для исправления нарушений плагин может предлагать быстрые исправления с использованием Quick Fix.

Выполнение проверок запускается автоматически при сохранении изменений пользователем.

Листинг 4 – Псевдокод варианта реализации этапа Develop

```
override fun documentChanged(event: DocumentEvent) {
    val file = event.document.virtualFile
    if (isSourceCode(file)) {
        val issues = runCodeStyleCheck(file)
        if (issues.isNotEmpty()) {
            notifyUser("Найдены проблемы с код-стайлом: $issues")
        }
    }
}
```

На этапе Develop плагин также может предоставлять пользователю подсказки по рефакторингу или улучшению производительности кода, активируя их по запросу через соответствующее меню.

Этап Control начинается, когда разработчик завершает редактирование функциональности и инициирует процесс тестирования. Триггером для этого этапа служит сохранение всех изменений и запуск тестов вручную через плагин или IDE.

Результаты тестов обрабатываются плагином и отображаются пользователю в структурированном виде. В случае успешного выполнения всех тестов плагин предлагает создать commit и выполнить push изменения в удаленный репозиторий для дальнейшего создания PR и прохождения CR. Если тесты завершились с ошибками, плагин уведомляет пользователя и предоставляет детальный отчет с указанием причин их падения. Пример инициации тестов выглядит следующим образом:

Листинг 5 – Псевдокод варианта реализации этапа Control

```
val testManager = TestManager.getInstance(project)
testManager.runAllTests {
    if (it.isSuccess) {
        notifyUser("Все тесты пройдены. Создайте Pull Request")
    } else {
        notifyUser("Ошибка тестов: ${it.errors}")
    }
}
```

На завершающем этапе Apply разработчик принимает решение об интеграции изменений в основную ветку. Триггером для перехода к этапу Apply является успешное прохождение код-ревью и отсутствие конфликтов при слиянии. Плагин отслеживает статус PR и, в случае положительного результата, предлагает выполнить слияние ветки в основную ветку. После завершения слияния плагин автоматически удаляет временную ветку и обновляет локальный репозиторий.

Процесс слияния может быть реализован следующим образом:

Листинг 6 – Псевдокод варианта реализации этапа Apply

```
val branchName = "feature/new-feature"
git.mergeBranch(project, "main", branchName)
git.deleteBranch(project, branchName)
notifyUser("Ветка $branchName успешно слита и удалена")
```

Описанная выше логика может стать основой активного режима работы плагина, в то время, как индикативный режим работы плагина будет отображать подсказки на основе текущего этапа выполнения методологии.

Таким образом, действия пользователя на каждом этапе методологии CODE становятся триггерами для выполнения автоматизированных операций, контролируемых плагином. Система не только направляет разработчика по методологии, но и минимизирует ошибки, повышая эффективность и качество разработки. Плагин, реализующий методологию CODE, сможет обеспечить последовательное и стандартизированное выполнение всех этапов разработки, повышая качество кода и эффективность работы команды. Следующим этапом является непосредственная разработка плагина с техническими подробностями, однако это выходит за рамки настоящей статьи.

Заключение

Итак, в данной работе мы рассмотрели концептуальный подход к интеграции методологии CODE в процесс разработки программного обеспечения посредством создания плагина для среды разработки. Было суждено, что практическая реализация методологии в виде инструмента, интегрированного в рабочую среду разработчиков, значительно упрощает её внедрение и использование. Подробно описаны этапы методологии (Prepare, Develop, Control и Apply) и логика работы плагина на каждом из них. Приведены примеры действий пользователя, инициирующих переход между этапами, и реакции системы, включая проверку стандартов именования веток, соблюдение код-стайла, автоматизацию тестирования и управление процессом слияния изменений.

Рассмотренные аспекты продемонстрировали, как инструментальная реализация методологии может повысить стандартизацию, удобство и качество процессов разработки. Плагин предлагает последовательное выполнение этапов методологии, минимизирует влияние человеческого фактора и способствует улучшению коллективной работы.

Перспективность развития методологии обусловлена в том числе запуском нового национального проекта «Экономика данных», пришедшим на смену предыдущему проекту «Цифровая экономика» [21]. Национальный проект «Цифровая экономика» был рассчитан на 6 лет, и 10 января 2025 года были подведены его итоги [22]. Нормативное регулирование цифровой среды было одной из целей предыдущего этапа развития цифровизации, на новом этапе выделяются направления развития стандартов работы с данными, а также обработки и анализа данных, репозитории открытого кода [23], что свидетельствует о высоком интересе на уровне государства к сфере формализации информационных технологий, в том числе управления кодом как данными.

Продолжением данной работы станет техническая реализация описанного плагина, включая разработку пользовательских интерфейсов, интеграцию с системой контроля версий и реализацию функций для каждого этапа методологии. Технические аспекты разработки, включая кодовые решения, архитектуру плагина и детали его тестирования, будут подробно освещены в следующем исследовании. Таким образом, предложенная концепция переходит в стадию реализации, что станет важным шагом к практическому применению методологии CODE в реальных проектах.

Литература

1. Зекирьяев, Р. Т. Методология управления кодовой базой программных комплексов в условиях цифровой экономики / Р. Т. Зекирьяев, Р. Г. Болбаков // Цифровая экономика. – 2024. – № 1(27). – С. 78-85.
2. Deming, W. Edwards (2000). Out of the crisis (1. MIT Press ed.). Cambridge, Mass.: MIT Press, 2000. p. 88. ISBN 0262541157.
3. Coding conventions. URL: <https://kotlinlang.org/docs/coding-conventions.html>
4. Baum, Tobias; Liskin, Olga; Niklas, Kai; Schneider, Kurt (2016). "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes". 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 74–85.
5. What is an IDE? URL: <https://aws.amazon.com/what-is/ide>
6. IDE (Integrated Development Environment) Plugins. URL: <https://startup-house.com/glossary/ide-integrated-development-environment-plugins>
7. Getting started | IntelliJ IDEA Documentation URL: <https://www.jetbrains.com/help/idea/getting-started.html>
8. IntelliJ Platform SDK. URL: <https://plugins.jetbrains.com/docs/intellij/welcome.html>
9. Hotfix. URL: <https://en.wikipedia.org/wiki/Hotfix>
10. Sward, David. Measuring the Business Value of Information Technology Practical Strategies for IT and Business Managers. Intell Press. 2006. 282 p.
11. Marco Ortu, Giuseppe Destefanis, Daniel Graziotin, Michele Marchesi, Roberto Tonelli. How Do You Propose Your Code Changes? Empirical Analysis of Affect Metrics of Pull Requests on GitHub. 2020. 11 p.
12. Amanda J. Painter, Brenda A. Haire. The Onboarding Process: How to Connect Your New Hire. 2022.
13. Mike Terhar. GitLab Book: Git Branching Strategies 2020. URL: <https://brownfield.dev/post/2020-06-27-git-branching-book/>
14. Ryan Dewhurst. Static Code Analysis. URL: https://owasp.org/www-community/controls/Static_Code_Analysis
15. Gradle User Manual. URL: <https://docs.gradle.org/current/userguide/userguide.html>
16. Android Studio. URL: <https://developer.android.com/studio>
17. How Many Software Engineers Are There in the World? URL: <https://www.allstarsit.com/blog/how-many-software-engineers-are-there-in-the-world>
18. Maven Central. URL: <https://mvnrepository.com/>
19. SDK Platform Tools release notes. URL: <https://developer.android.com/tools/releases/platform-tools>
20. Scott Chacon, Ben Straub. Pro Git. New York, NY: Apress. 2025. 495 p.
21. «Цифровая экономика РФ»: Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации. URL: <https://digital.gov.ru/ru/activity/directions/858/>

22. Нацпроект «Цифровая экономика»: Результаты за 6 лет. URL: <https://национальныепроекты.рф/news/natsproekt-tsifrovaya-ekonomika-rezultaty-za-6-let/>
23. В России появится новый нацпроект — «Экономика данных»: Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации. URL: <https://digital.gov.ru/ru/events/45686/>.

References in Cyrillics

1. Zekir'yaev, R. T. Metodologiya upravleniya kodovoj bazoj programmnyh kompleksov v usloviyah cifrovoj ekonomiki / R. T. Zekir'yaev, R. G. Bolbakov // Cifrovaya ekonomika. – 2024. – № 1(27). – S. 78-85.
2. Deming, W. Edwards (2000). Out of the crisis (1. MIT Press ed.). Cambridge, Mass.: MIT Press, 2000. p. 88. ISBN 0262541157.
3. Coding conventions. URL: <https://kotlinlang.org/docs/coding-conventions.html>
4. Baum, Tobias; Liskin, Olga; Niklas, Kai; Schneider, Kurt (2016). "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes". 2016 IEEE International Conference on Software Quality, Reliability and Security (QRS). pp. 74–85.
5. What is an IDE? URL: <https://aws.amazon.com/what-is/ide>
6. IDE (Integrated Development Environment) Plugins. URL: <https://startup-house.com/glossary/ide-integrated-development-environment-plugins>
7. Getting started | IntelliJ IDEA Documentation URL: <https://www.jetbrains.com/help/idea/getting-started.html>
8. IntelliJ Platform SDK. URL: <https://plugins.jetbrains.com/docs/intellij/welcome.html>
9. Hotfix. URL: <https://en.wikipedia.org/wiki/Hotfix>
10. Sward, David. Measuring the Business Value of Information Technology Practical Strategies for IT and Business Managers. Intell Press. 2006. 282 p.
11. Marco Ortu, Giuseppe Destefanis, Daniel Graziotin, Michele Marchesi, Roberto Tonelli. How Do You Propose Your Code Changes? Empirical Analysis of Affect Metrics of Pull Requests on GitHub. 2020. 11 p.
12. Amanda J. Painter, Brenda A. Haire. The Onboarding Process: How to Connect Your New Hire. 2022.
13. Mike Terhar. GitLab Book: Git Branching Strategies 2020. URL: <https://brownfield.dev/post/2020-06-27-git-branching-book/>
14. Ryan Dewhurst. Static Code Analysis. URL: https://owasp.org/www-community/controls/Static_Code_Analysis
15. Gradle User Manual. URL: <https://docs.gradle.org/current/userguide/userguide.html>
16. Android Studio. URL: <https://developer.android.com/studio>
17. How Many Software Engineers Are There in the World? URL: <https://www.allstarsit.com/blog/how-many-software-engineers-are-there-in-the-world>
18. Maven Central. URL: <https://mvnrepository.com/>
19. SDK Platform Tools release notes. URL: <https://developer.android.com/tools/releases/platform-tools>
20. Scott Chacon, Ben Straub. Pro Git. New York, NY: Apress. 2025. 495 p.
21. «Cifrovaya ekonomika RF»: Ministerstvo cifrovogo razvitiya, svyazi i massovyh kommunikacij Rossijskoj Federacii. URL: <https://digital.gov.ru/ru/activity/directions/858/>
22. Nacproekt «Cifrovaya ekonomika»: Rezul'taty za 6 let. URL: <https://национальныепроекты.рф/news/natsproekt-tsifrovaya-ekonomika-rezultaty-za-6-let/>
23. V Rossii poyavitsya novyj nacproekt — «Ekonomika dannyh»: Ministerstvo cifrovogo razvitiya, svyazi i massovyh kommunikacij Rossijskoj Federacii. URL: <https://digital.gov.ru/ru/events/45686/>.

*Зекирьяев Руслан Тимурович,
аспирант кафедры инструментального и прикладного программного обеспечения
Института информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: auriax@ya.ru
ORCID: 0000-0003-3787-7511*

*Болбаков Роман Геннадьевич, к.т.н., доцент,
заведующий кафедрой инструментального и прикладного программного обеспечения Института
информационных технологий ФГБОУ ВО «МИРЭА – Российский технологический университет»
(119454, Россия, Москва, пр-т Вернадского, д. 78).
E-mail: bolbakov@mirea.ru. ORCID: 0000-0002-4922-7260*

Ключевые слова

Разработка программного обеспечения, управление разработкой программного обеспечения, кодовая база, разработка плагина, методология CODE

Ruslan T. Zekiriaev, Roman G. Bolbakov. Practical implementation of the CODE methodology

Keywords

Software Development, Software Project Management, Code Base, Information Technology Standards, Plugin Development, CODE Methodology.

DOI: 10.34706/DE-2025-01-02

JEL classification O32.

Abstract

The article deals with the integration of the CODE (Codebase Operations and Development Practices) methodology into the software development process through the creation of a plug-in for the IntelliJ IDEA integrated development environment. The CODE methodology, based on the Deming-Schuhart cycle adaptation (PDCA), includes four stages: Prepare, Develop, Control and Apply, each of which is aimed at standardization and automation of codebase management.

The paper substantiates the necessity of practical implementation of the methodology in the form of a tool integrated into the development environment. It is shown that creating a plug-in for the development environment allows developers to effectively comply with standards, improve code quality and speed up the performance of everyday tasks. The article describes the logic of the plugin, key functions at each stage of the methodology and examples of user-initiated actions such as creating new branches, checking code compliance, running tests and managing the process of merging changes.

Special attention is given to the initial stage of plugin development, including project settings and using the IntelliJ IDEA API to interact with version control systems, code analysis, and testing. Conceptual code samples are provided to illustrate the implementation of functionality.

The aspects discussed demonstrate the potential of using a plug-in to simplify the implementation of the CODE methodology into real-world team development practices. A direct technical implementation of the plug-in could be a continuation of this research. The presented ideas lay the foundation for practical application of the CODE methodology aimed at increasing the efficiency and standardization of development processes.

УДК: 004.43, 004.4'4

1.3. Разработка языка Тривиль. Часть 3. Баланс

А. Е. Недоря, г. Санкт-Петербург, Россия

Статья является третьей из серии статей, в которых описывается разработка языка программирования Тривиль. В предыдущих статьях были определены цели языка, требования к языку и описаны основные языковые решения. Эта статья рассматривает сложные языковые конструкции, в разработке которых основное внимание уделялось балансу между полнотой языковой конструкции и сложностью языка и компилятора. Как и вся серия, статья нацелена, большей частью, не на программиста, который использует язык, а на разработчика языков программирования.

Введение

В первых двух статьях серии [1, 2] были определены цели языка, основные требования к языку и описаны основные конструкции языка Тривиль, при этом особое внимание было уделено тому, как выбор конструкций вытекал из целей и требований. В данной статье будет рассмотрено несколько сложных конструкций, в разработке которых приходилось искать баланс между полнотой конструкции и сложностью ее реализации.

Содержание этой статьи можно кратко передать фразой: **“Лучше день потерять, а потом за пять минут долететь” или нет?**

Фраза содержит цитату из известного советского мультфильма¹, в котором гриф пытался научить страуса летать, чтобы добраться до вкусного за 5 минут полета вместо долгого бега.

В разработке Тривилия я неоднократно сталкивался с выбором такого же типа:

- если в язык добавить некоторую конструкцию, то она упростит дальнейшую разработку и **ускорит** переход к разработке следующих языков семейства (лучше день потерять);
- в то же время разработка языковой конструкции и реализация её в компиляторе может потребовать много усилий, а это **замедлит** переход к разработке следующих языков.

Может показаться, что это редкая ситуация. Увы, разработчики программных систем постоянно вынуждены делать выборы примерно такого типа. Даже если они понимают, как сделать правильно, хорошо и надолго (что далеко не всегда), обстоятельства часто вынуждают использовать *моток синей изоленты* вместо хорошего решения и надеяться, что когда-нибудь это место будет переделано с надлежащим качеством. Для такой временки даже есть специальный термин: *технический долг*.

Любопытно, что технический долг — это специфика программной инженерии. Почему? Потому что недооценка времени и трудозатрат, задержка времени выполнения проектов и прочие неприятные вещи являются обязательными спутниками (почти) всех программных проектов. В одной из следующих статей мы подробно поговорим о том, почему это происходит, кто виноват и что делать. Впрочем, ответ на вопрос *кто виноват?* можно дать прямо сейчас — виноваты мы, а именно: “товарищи ученые, доценты с кандидатами”².

Вернусь к Тривилию. Часть конструкций языка я считаю обязательными, независимо от затрат на их реализацию, например, те, что реализуют безопасность ссылок (null safety). Часть — такими, без которых можно обойтись, хоть это может быть в чем-то неудобно. Для части конструкций (полное решение которых трудозатратно) приходилось искать компромиссные решения, которые покрывают существенную часть необходимой функциональности, но, в то же время, могут быть сделаны за небольшое время.

В следующих разделах мы рассмотрим такие вопросы:

- операторы цикла;
- инициализация объектов;
- реализация обобщенных хеш-таблиц.

Циклы всякие нужны

В предыдущей статье в списке операторов был указан единственный оператор цикла — цикл пока. И это понятно: так вместе с оператором прервать (break) этот цикл позволяет выразить любое повторяющееся действие. Зачем же в языки программирования добавляют другие операторы цикла? Для увеличения выразительности и для уменьшения числа ошибок (что равно энергосбережению). Например, если обработку элемента массива делать циклом пока, то можно ошибиться в проверке условия и изменении счетчика, да и запись становится сложнее.

Рассмотрим, какие циклы есть в других языках на примере Go и затем попробуем найти баланс между усложнением языка и компилятора и упрощением программирования.

¹ Мультфильм “Крылья, ноги и хвосты”, 1987 г.

² В. Высоцкий, “Товарищи ученые”

1	бесконечный цикл	<code>for { ... break ... }</code>
2	цикл с предусловием	<code>for a < b { ... }</code>
3	арифметический цикл	<code>for i := 0; i < 10; i++ { ... }</code>
4	цикл перебора элементов	<code>for i, val := range arr { ... }</code>

Бесконечный цикл легко выражается через цикл с предусловием, арифметический цикл не имеет смысла в компиляторах, а вот перебор элементов массива или контейнера — это нужное действие. Цикл перебора элементов часто называется `foreach` и есть во всех современных языках программирования, см., например, Kotlin, Swift, Rust.

В общем виде цикл позволяет перебирать элементы любого контейнера и требует, чтобы язык поддерживал:

- итераторы, чтобы разработчик библиотеки контейнера мог обеспечить возможность использования цикла;
- возможность возвращать из итератора несколько значений, например, через поддержку кортежей (Swift) или функций, возвращающих несколько значений (Go).

И то, и другое существенно влияет и увеличивает сложность языка и компилятора и в общем виде не подходит для Тривилия.

Что именно нужно перебирать в компиляторе? Элементы массива и элементы словаря (`hash-map`). Увы, перебор элементов словаря придется отложить на следующий не-совсем-тривиль язык, и как мы дальше увидим, это вполне оправдано.

Рассмотрим подробнее перебор элементов массива. Есть три варианта того, что используется в теле цикла:

- значение элемента;
- индекс элемента;
- индекс и значение.

Go поддерживает все три варианта за счет использования символа игнорирования:

- нужно только значение: `for _, val := range arr {...}`;
- нужен только индекс `for i, _ := range arr {...}` или `for i := range arr {...}`.

Вот последняя запись меня удивляет: на мой взгляд, значение “важнее” и чаще используется, почему же запись с одним индексом короче? Когда я пишу на Go, я каждый раз теряю время/силы на то, чтобы вспомнить, как надо писать, пусть совсем немного, но теряю. Поэтому я приложил существенные усилия, чтобы найти синтаксис, который более очевиден и, следовательно, сберегает энергию. Вот что получилось:

Что перебирается	Go	Тривиль	Исп. (*)
массив: только значение	<code>for _, val := range arr {...}</code>	цикл эл среди arr {...}	137
массив: индекс, значение	<code>for i, val := range arr {...}</code>	цикл [№]эл среди arr {...}	16
массив: только индекс	<code>for i := range arr {...}</code>	цикл [№] среди arr {...}	0
словарь: ключ, значение	<code>for key, val := range hm {...}</code>	нет поддерживается	47

В последнем столбце записано число использований конструкции в компиляторе Тривилия. Как видно, перебор значений — это самая используемая конструкция и она самая короткая в Тривиле. Если в цикле нужен индекс, используется дополнительный синтаксис, указывающий, что это именно индекс, то есть то, что используется для индексации. А вот перебор только индексов, то есть то, что в Go записывается короче всего, вообще не используется в компиляторе. Перебор элементов словаря в Тривиле делается отдельным образом, без использования такого цикла.

Важно то, что перебор элементов массива — это $\frac{3}{4}$ всех случаев перебора элементов в компиляторе (153 из 200). Выбранное частное решение хорошо тем, что

- покрывает существенную часть случаев,
- не требует добавления в язык сложных конструкций,
- и является простым в реализации.

Инициализация объектов

Одно из сложных мест в любом языке, который каким-либо образом поддерживает ООП — это инициализация объектов. Это сущностно сложная конструкция, поэтому особые усилия должны быть направлены на ее упрощение в языке.

Инициализация объекта должна привести объект в *корректное* состояние, или состояние готовности к работе, после чего любые *корректные* действия над объектом должны сохранять корректное состояние (другими словами, сохранять инвариант класса или объекта³).

Некоторые языки, такие как Swift, явно выделяют две фазы инициализации объекта [3], задача которых:

1. задать определенные значения полям объекта, так чтобы избежать неопределенного поведения (undefined behaviour);
2. перевести объект в семантически *корректное* состояние или прервать выполнение, если это невозможно.

На мой взгляд, выделение двух фаз — это верный подход, так как язык не может гарантировать корректность семантики (это задача разработчика), даже если инвариант явно задан⁴ (см. Eiffel). А вот гарантировать, что все поля проинициализированы, язык и компилятор могут и обязаны делать в языке, который заявлен как безопасный⁵.

Далее в статье будут рассмотрены подходы к инициализации в Swift, Kotlin и Go, потом в Тривиле и, далее рассмотрены ограничения Тривилия: баланс сложности (простоты) и полноты.

Для сравнения сначала определим критерии или требования:

1. безопасность: инициализация должна выполняться безопасно, то есть язык и компилятор должны гарантировать отсутствие ошибок в первой фазе. Вторая фаза — в зоне ответственности разработчика;
2. простота: инициализация должна записываться просто, насколько возможно, но, обязательно безопасно, см. п.1;
3. полнота: все “нормальные” (безопасные) случаи должны быть поддержаны.

Инициализация в Swift

Рассмотрим пример, в котором есть класс, каждый объект которого содержит поле с порядковым номером объекта (в порядке создания):

```
var counter = 0 // счетчик объектов

class Counted {
    var ord: Int
    init() {
        // фаза 1:
        self.ord = 0
        // фаза 2:
        counter += 1
        self.ord = counter
    }
}

var first = Counted()
print(first.ord)
```

Выполнение примера выводит 1 на консоль.

³ Для языков, в которых нет классов: ООП, да не CLOP.

⁴ Это никак не относится к языкам, в которых неопределенное состояние является штатным.

⁵ CLOP язык — язык поддерживающий class-oriented programming.

Как мы видим, на фазе 1 происходит инициализация всех полей класса (одного, в данном случае), и она заканчивается, когда все поля проинициализированы. Фаза 2 переводит объект в семантически корректное состояние. В нашем примере, это состояние, когда значение поля `ord` — это порядковый номер объекта.

Код инициализации в примере намеренно сделан избыточным, чтобы показать разделение на фазы. Его можно переписать так, чтобы вся инициализация проходила на фазе 1:

```
init() {
  // фаза 1
  counter += 1
  self.ord = counter
  // начало фазы 2
}
```

Во время фазы 1 нельзя обратиться к полям или методам объекта, например, нельзя обратиться к полю до его инициализации:

```
init() {
  // фаза 1
  print(self.ord) // error - used before being initialized
  counter += 1
  self.ord = counter
  // начало фазы 2
}
```

Компилятор покажет ошибку. Отсутствие инициализации тоже приведет к сообщению об ошибке:

```
class Recursive {
  var next: Recursive // error - not initialized
}
```

А такой код скомпилируется, но приведет к бесконечной рекурсии во время выполнения, что более-менее приемлемо, так как соответствует поведению для обычных рекурсивных функций:

```
class Recursive {
  var next: Recursive
  init() {
    next = Recursive()
  }
}
```

У внимательного читателя, незнакомого с Swift, может возникнуть вопрос, как описывать рекурсивные структуры, например, дерево. Ответом является использование опционального типа:

```
class Tree {
  var l: Tree? = Optional.none
  var r: Tree? = Optional.none
}
```

В этих примерах показана малая часть того, что есть в инициализации объектов в Swift, размер раздела инициализации в спецификации языка — это около 30 страниц текста. Но нам пока достаточно, чтобы сравнить с другими языками.

Замечу, что часть кода в примерах можно было написать короче, например, опустить `self` или писать `nil` вместо `Optional.none`. Я намеренно использую более полный и ясный для читателя вариант кода.

Краткий вывод для Тривиля: очень сложно, не надо так.

Инициализация в Kotlin

Повторим пример с порядковым номером объекта:

```
var counter = 0

class Counted {
    var ord: Int
    init {
        counter += 1
        this.ord = counter
    }
}

fun main() {
    val first = Counted()
    print(first.ord)
}
```

Код выглядит очень похоже на код на Swift и ведет себя также. В примере с печатью неинициализированного поля компилятор также сообщает об ошибке, а пример с рекурсивным типом компилируется и приводит к бесконечной рекурсии при выполнении. В примере с деревом также надо использовать опциональный тип.

А вот следующий пример показывает принципиальную разницу между двумя языками:

	Kotlin		Swift
1	<code>class A { val f = 1 }</code>	1	<code>class A { var f = 1 }</code>
2		2	
3	<code>class C {</code>	3	<code>class C {</code>
4	<code>val a: A</code>	4	<code>var a: A</code>
5	<code>init {</code>	5	<code>init() {</code>
6	<code>fn()</code>	6	<code>fn() // error</code>
7	<code>a = A()</code>	7	<code>a = A()</code>
8	<code>}</code>	8	<code>}</code>
9	<code>public fun fn() {</code>	9	<code>public func fn() { }</code>
10	<code>println(a.f)</code>	10	<code>}</code>
11	<code>}</code>		
12	<code>}</code>		
13	<code>fun main() {</code>		
14	<code>val c = C()</code>		
15	<code>c.fn()</code>		
16	<code>}</code>		

Пример на Kotlin успешно компилируется, а при **выполнении** выдает ошибку в строке 10: Cannot invoke "A.getF()" because "this.a" is null.

А в примере на Swift ошибку выдает **компилятор** в строке 6: вызов метода объекта до инициализации объекта.

Разница в том, что Kotlin заявлен как язык с безопасными ссылками (null safe), а в примере не проверяется доступ к неинициализированной переменной, что недопустимо в таком языке. Подробнее об этом можно прочитать в [4].

Краткий вывод для Тривилья: сложно, да еще и небезопасно. Не надо так.

Инициализация в Go

Инициализация объектов в Go существенно отличается от инициализации в Swift и Kotlin, так как

- язык не поддерживает безопасность ссылок (not null safe);
- нет классов, классы эмулируются указателями на структуры;
- нет инициализирующих методов;
- для создания объектов используются композитные литералы (composite literals).

Пример с порядковым номером объекта:

```

1 package main
2 import "fmt"
3
4 var counter = 0
5
6 type Counted = struct {
7     ord int
8 }
9
10 func New() *Counted {
11     counter++
12     return &Counted{ord: counter}
13 }
14
15 func main() {
16     var first = New()
17     fmt.Println(first.ord)
18 }

```

Текст выглядит непривычно для тех, кто привык писать на CLOP⁶ языках, поэтому несколько пояснений:

1. первая фаза инициализации в Go выполняется автоматически, так как для переменной любого типа есть значение по умолчанию (0 для int, nil для указателей и т.д.). Кажется, что это хорошо, но это “хорошее” свойство является следствием небезопасных решений, а именно:
 - в языке есть явные указатели;
 - любой указатель может быть “пустым”, то есть иметь значение null⁷.
2. конструкторов в языке нет, поэтому вторую фазу инициализации приходится делать через отдельную функцию (строки 10-13);
3. в строке 12 записан композитный литерал: Counted{ord: counter}, который задает значения полей структуры, затем к литералу применяется префиксный оператор &, который должен выдать адрес структуры. При этом срабатывает алгоритм escape-анализа [5], который копирует структуру в кучу, выдавая адрес выделенной в куче памяти.

Рассматривать другие примеры нет смысла, они не добавляют качества.

Краткий вывод для Тривилья: сложно, использует неявные алгоритмы, нельзя напрямую использовать, так как язык существенно другой.

⁶ См. Tony Hoar's billion-dollar mistake.

⁷ Я стараюсь не использовать транслитерацию английских терминов, но в данном случае, я не знаю лучшего решения. Термин “обобщенные типы и функции” слишком длинный и не очень удачный. Так что следом за другими переводами, я использую термин “джереник”.

Инициализация в Тривиле

Наш небольшой экскурс в сравнительное изучение инициализации объектов привел нас к тому, что выбирать не из чего, точнее, все варианты плохие:

- подход Swift: обеспечивает безопасность, но очень сложный;
- подход Kotlin: не обеспечивает безопасность и очень сложный;
- подход Go: не обеспечивает безопасность, основан на неявной семантике.

Придется делать свое. Вспомним требования к инициализации, которые были выписаны в начале раздела:

- безопасность,
- простота,
- полнота.

Начнем с простоты, потому что здесь возможны компромиссы, а в обеспечении безопасности — нет. Подход Go существенно проще. Возможно, что это не видно из статьи, так как я не рассматриваю сложные случаи, но хорошо заметно, если сравнить длину раздела в спецификациях языков Go и Swift.

За счет чего сделано упрощение?

За счет использования декларативного языка⁸ для создания объекта вместо императивного. В Go инициализация объекта — это всего лишь задание значений полей. В Swift (и во многих других языках) для инициализации используется специальная функция (конструктор или инициализатор). При этом в Go, пока объект не создан, его нельзя использовать, просто потому что нет никакого *магического* идентификатора, типа `self` или `this`. Соответственно, нет лишних языковых конструкций и нет стоящей за ними сложной семантики.

Попробуем применить подход к Тривилю (английские идентификаторы оставлены намеренно для упрощения сравнения):

	Тривиль		Go
1	модуль <code>main</code>	1	<code>package main</code>
2	импорт <code>"std::вывод"</code>	2	<code>import "fmt"</code>
3		3	
4	пусть <code>counter := 0</code>	4	<code>var counter = 0</code>
5		5	
6	тип <code>Counted = класс</code> {	6	<code>type Counted = struct</code> {
7	<code>ord := 0</code>	7	<code>ord int</code>
8	}	8	}
9		9	
10	фн <code>New(): Counted</code> {	10	<code>func New() *Counted</code> {
11	<code>counter++</code>	11	<code>counter++</code>
12	вернуть <code>Counted{ord: counter}</code>	12	<code>return &Counted{ord: counter}</code>
13	}	13	}
14		14	
15	вход {	15	<code>func main() {</code>
16	пусть <code>first = New()</code>	16	<code>var first = New()</code>
17	вывод . <code>ф("\$:\n", first.ord)</code>	17	<code>fmt.Println(first.ord)</code>
18	}	18	}

Текст очень похож, хотя есть два существенных отличия, как синтаксических, так и семантических:

1. Тривиль обеспечивает выполнение первой фазы инициализации за счет обязательной явной инициализации полей (строка 7), а не за счет инициализации по умолчанию. Инициализация поля может быть отложена, но она всегда явная.
2. Конструктор класса в Тривиле (строка 12) сразу же создает объект класса, здесь нет неявной семантики (нет `escape`-анализа).

Обеспечивается ли в Тривиле безопасность? Да, обеспечивается, покажем на следующих фрагментах кода.

⁸ Make it simple as possible, but not simpler — слова А.Эйнштейна, использованные Никлаусом Виртом как лого языка Oberon.

Пример 1. Отсутствие инициализации поля.

В Тривиле инициализация полей обязательна, но ее можно отложить, используя ключевое слово **позже**, но только до конструктора класса. Если в конструкторе класса поле не задано, то компилятор выдаст ошибку:

```
тип К = класс {
    номер: Цел64 = позже
}
пусть к = К{} // ошибка: значение поле 'номер' должно быть задано
```

Замечу, что семантическая проверка в данном случае тривиальна, в отличие от того, что приходится делать в компиляторах Swift и Kotlin.

Пример 2. Рекурсивный тип. В отличие от Swift и Kotlin, где программа падает по бесконечной рекурсии, компилятор Тривилия выдает ошибку:

```
тип К = класс {
    номер: Цел64 = позже
}
пусть к = К{} // ошибка: значение поле 'номер' должно быть задано }
```

Эту ошибку ловит общий алгоритм, отслеживающий рекурсивные определения, например, рекурсивное определение констант:

```
конст А = В // ошибка: рекурсивное определение 'А'
конст В = А
```

В итоге существенно более простой компилятор оказывается мощнее семантически.

Пример 3. Пример, который обнаруживает дыру в безопасности Kotlin, в Тривиле нельзя написать. Приведу текст на Swift (находит ошибку) и Тривиле (нельзя сделать ошибку):

	Swift		Тривиль
1	<code>class A { var f = 1 }</code>	1	<code>тип А = класс { ф := 1 }</code>
2		2	
3	<code>class C {</code>	3	<code>тип К = класс {</code>
4	<code> var a: A</code>	4	<code> а: А = позже</code>
5	<code> init() {</code>	5	<code>}</code>
6	<code> self.fn() // error</code>	6	
7	<code> a = A()</code>	7	<code>фн (к: К) Ф() {}</code>
8	<code> }</code>	8	
9	<code> public func fn() { }</code>	9	<code>пусть к = К{а: А{}}</code>
10	<code>}</code>	10	<code>к.Ф()</code>

В Тривиле нет возможности обратиться к недостроенному объекту (до завершения конструктора класса), поэтому нет необходимости в семантической проверке, которая совсем не тривиальна. Идеальный способ решения проблемы: нет проблемы, не надо решать.

Итак, из 3-х требований к инициализации, два (безопасность и простота) в Тривиле выполняются. А вот с полнотой есть проблема.

Проблема полноты инициализации в Тривиле

Рассмотрим пример с наследованием (в терминологии ООП):

- в модуле base определяется базовый класс А со счетчиком,
- в модуле main определяется класс В, расширяющий А, а счетчик из базового класса инициализируется начальным значением.

Напишем сначала код на Go:

1	<code>package base</code>	1	<code>package main</code>
2		2	<code>import "fmt"</code>
3	<code>type A struct {</code>	3	<code>import "base"</code>
4	<code> count int</code>	4	
5	<code>}</code>	5	<code>type B struct { base.A }</code>
6		6	
7	<code>func (a *A) Get() int {</code>	7	<code>func New(x int) *B {</code>
8	<code> return a.count</code>	8	<code> return &B{A: base.Init(x)}</code>
9	<code>}</code>	9	<code>}</code>
10		10	
11	<code>func Init(x int) A {</code>	11	<code>func main() {</code>
12	<code> return A{count: x}</code>	12	<code> fmt.Println(New(3).Get())</code>
13	<code>}</code>	13	<code>}</code>

В Go для того, чтобы имя могло быть использовано за пределами модуля (другими словами — экспортировано), оно должно начинаться с заглавной буквы. Имена `A`, `Get`, `Init` экспортируются из пакета `base`, а `count` (имя поля) не экспортируется. Счетчик скрыт (инкапсулирован) в пакете `base`. Для того, чтобы значение счетчика можно было задать, добавлена функция `Init`.

В пакете `main` описан тип `B`, содержащий тип `A`. Это, в некотором смысле, замена наследования, так как методы типа `A` могут быть применены к объекту типа `B`. (см. строку `main:12`). Функция `main` создает объект типа `*B` (указатель на структуру `B`).

Нас интересует строка `main:8`, где конструируется объект типа `B`, а значение под-объекта `A` задается функцией `base.Init`.

Попробуем теперь написать тот же пример на Тривиле:

1	<code>модуль base</code>	1	<code>модуль main</code>
2		2	<code>импорт "std::вывод"</code>
3	<code>тип A* = класс {</code>	3	<code>импорт "base"</code>
4	<code> count: Цел64 := позже</code>	4	
5	<code>}</code>	5	<code>тип B = класс (base.A) {}</code>
6		6	
7	<code>фн (a: A) Get*(): Цел64 {</code>	7	<code>фн New(x: Цел64): B {</code>
8	<code> вернуть a.count</code>	8	<code> вернуть B{count: x}</code>
9	<code>}</code>	9	<code>}</code>
10		10	
11	<code>фн Init*(x: Цел64): A {</code>	11	<code>вход {</code>
12	<code> вернуть A{count: x}</code>	12	<code> вывод.ф("\$\n", New(3).Get())</code>
13	<code>}</code>	13	<code>}</code>

В отличие от Go, в Тривиле экспорт явный (а не по заглавной букве), экспортируются имена, отмеченные символом `*`. То есть тип `A` экспортирован (и доступен из других модулей), а поле `count` — нет. И это приводит к проблемам.

Первая проблема видна уже в модуле `base`: функция `Init` возвращает указатель на объект (reference type), а не структуру (value type), так как в Тривиле нет структур, а это дополнительное выделение памяти⁹. Но основная проблема в модуле `main`, строка 8 — создать объект `B` невозможно, так как:

- поле `count` не экспортировано,
- для типа `B` использовано наследование, а не композиция, как в Go.

Наследование, как и в других языках, приводит к “плоской” модели, доступ к полю базового класса `A` не отличается от доступа к полю класса `B`. Кроме того, нет языковых средств, которые позволяют работать с полями базового класса, как с одним целым.

Как можно разрешить эту проблему?

- на уровне примера: экспортировать поле `count`, тогда код в строке `main:8` станет корректным, но будет потеряна инкапсуляция;
- на уровне языка: добавить некоторый языковый механизм, который решит эту проблему. Например, ввести понятие инициализатора, как в Swift (и сделать семантику языка существенно более сложной) или сделать что-то иное (что нетривиально, и, неизвестно, сколько займет времени и усилий).

⁹ Я рассматриваю очевидное решение, не претендуя на то, что оно лучшее и единственное.

На самом деле у меня есть наметки того, как решить эту задачу, не отказываясь от декларативного подхода, но я не собираюсь добавлять эту возможность в Тривиль. Просто потому, что в компиляторе не нужна возможность скрывать поле в базовом классе от наследников. Об инкапсуляции и о ложном её понимании мы подробнее поговорим в следующей статье. Пока же, решение простое — ничего не делать на уровне языка. В целом, это ограничение, но не мешающее двигаться дальше.

Взаимодействие конструкций

Задача инициализации объектов является хорошим примером, на котором можно показать одну из существенных сложностей в разработке языков, которая называется взаимодействие конструкций (feature interaction). Я упоминал о ней в первой статье, теперь есть материал для немного более подробного рассмотрения. Проблема эта широко известна в очень узком кругу разработчиков языков.

Вернемся к сравнительному анализу и перечислим конструкции, которые используются для инициализации объекта:

- Swift: функция инициализатор (initializer),
- Go: конструктор значения структуры (composite literal),
- Тривиль: конструктор значения объекта (ссылки).

Зададим вот такой вопрос — какие конструкции языка нам пришлось учитывать, думая про инициализацию объектов?

Конструкция	Swift	Go	Тривиль
Безопасность ссылок	есть	нет	есть
Наследование	есть	нет, композиция	есть
Значения полей по умолчанию	нет	есть	нет
Обязательная инициализация полей	нет	нет	есть
Видимость в других модулях	public, protected, private	экспорт	экспорт

Как показывает таблица, на инициализацию влияют, по крайней мере, 5 других языковых конструкций. Это обычная проблема в разработке языка, которая хорошо описывается поговоркой: хвост вытащил, нос увяз.

И еще одно замечание, как видно из таблицы, нет ни одной строки, в которой все три языка используют одинаковое решение. Решение не устоялось, идет активный поиск.

Что нам стоит хеш-таблицу построить?

Я уже упоминал в [2] о том, что для разработки компиляторов нужен тип hash-map, точнее, нужен контейнер, который я называю Словарь, который хранит пары (ключ, значение) и обеспечивает быстрый доступ по ключу. Называть этот контейнер hash-map не совсем корректно, так как это название способа реализации, который, теоретически, может быть разным.

Впрочем, это традиционное название: в Rust и Kotlin такой контейнер называется HashMap, в Go — map, а вот Swift пошел другим путем и назвал его — Dictionary.

С точки зрения реализации, есть существенная разница между Go и остальными языками, в Go map — это встроенный тип, определенный в языке, в остальных языках он реализован в библиотеке. Об этом мы поговорим немного позже.

Важность типа Словарь для компилятора подтверждает статистика: В компиляторе Тривилия тип Словарь используется 23 раза, причем с разными комбинациями типов (ключ, значение). Число комбинаций слишком велико, чтобы писать реализацию каждой комбинации отдельно. Чтобы избежать дублирования кода, есть два пути:

- путь Go: встроить тип в язык;
- путь остальных языков: использовать обобщенные типы или дженерики¹⁰ (generics).

Замечу, что в Go дженерики появились в 2022 году (версия 1.18), через 10 лет после выхода первой версии языка. Полагаю, что, если бы они были добавлены в Go изначально, тип map тоже был бы библиотечным.

Дженерики позволяют писать полиморфный код, то есть код, который работает с множеством наборов типов, а это экономия на разработке, отладке и сопровождении. Дженерик — это языковая конструкция, обычно тип или функция, параметризованная типами (типовыми параметрами). Хороший обзор дженериков сделан в статье [6].

¹⁰ Ошибка, например, может быть в коде проверки инварианта.

Замечу, что дженерики — это далеко не единственный способ писать полиморфный код, и это далеко не *серебряная пуля*, но разговор об этом не входит в тему статьи.

С использованием дженериков в библиотеке можно написать (на некотором языке) тип `Map<K, V>` который настраивается на конкретные типы, например, `Map<string, int>`.

Вернемся к Тривилю. Что выбрать из двух вариантов: добавление встроенного типа или добавление дженериков? Увы, это выбор между сложным и частным (встроенный тип) и более универсальным и очень сложным (дженерики).

Идти по пути Go мне представляется неверным, добавление еще одного конструируемого типа к трем уже имеющимся увеличивает сложность языка и компилятора, а реализацию этого типа придется писать в коде поддержки исполнения (runtime support) на другом языке программирования, что увеличивает сложность разработки и отладки. Так что этот путь я отбросил сразу.

Подумаем о добавлении дженериков в язык. На опыте разработки и реализации других языков я знаю, что добавление в язык дженериков — это очень непростая задача, как с точки зрения самого языка (синтаксиса и, особенно, семантики), так и реализации в компиляторе.

Так как путь Тривиля — это путь упрощения (make it as simple as possible), то подумаем, можем ли мы предложить некое упрощение дженериков, уменьшающее объем работы, но все же позволяющее написать Словарь (but not simpler¹¹). Для того, чтобы понять, что можно упростить, надо определить, в чем сложность дженериков. Не претендуя на полноту, перечислю три проблемы:

- сложность семантики,
- необходимость введения в язык ограничений на типовые параметры (constraints),
- баланс между производительностью и размером кода.

Проблема 1. Сложность семантики

Не вдаваясь в детали, скажу лишь о том, что дженерик-типы добавляют в язык три новых категории типов со своим семантическими правилами:

- обобщенные типы, например, `Map<K, V>`,
- конкретные типы, например, `Map<string, Person>`,
- частично-конкретные типы, например, `Map<string, V>`.

При этом семантические правила для каждого из этих типов обычно весьма нетривиальны. Любопытно, что только конкретные типы существуют во время исполнения программы, обобщенные и частично-конкретные типы — это абстракции, существующие только во время компиляции.

Проблема 2. Нужны ограничения на типовые параметры

Рассмотрим простой пример. Мы хотим написать функцию сортировки `sort<T>`, строящую упорядоченный список элементов контейнера. Для сортировки содержимого надо¹², чтобы

- для контейнера была определена операция получения числа элементов,
- и операция извлечения элемента, например, по порядковому номеру,
- для элементов была определена операция сравнения.

Очевидно, что любой тип не может быть использован вместо типа `T`. Для того, чтобы компилятор мог проверить семантическую корректность настройки функции, язык должен позволить задать ограничения (constraints) типового параметра, как в списке выше.

Ограничение может быть задано неким интерфейсом (Go interface, Swift protocol), назовем его `Sortable`. Часть проблем мы закрыли, но, например, мы хотим использовать функцию также для сортировки встроенных массивов, для которых интерфейс нельзя задать. Придется делать ограничения более сложными, и это может привести к необходимости переопределения операций (operator overloading), определения методов для любых типов языка (в том числе предопределенных) и к прочим веселым последствиям (например, unified type system), которые существенно усложняют язык и компилятор.

Проблема 3. Как найти баланс между производительностью и размером кода

Речь идет о том, сколько экземпляров кода будет создано для типов с разными типовыми аргументами, например: `Map<string, int>`, `Map<int, string>` и `Map<string, Person>`.

Два крайних подхода используют C++ и Java:

- C++ делает реплику кода для каждой настройки дженерика, что дает возможность увеличить производительность за счет оптимизаций кода. Например, если в коде дженерик-функции используется знак "+", то в настройке этой функции на целый тип будет целое сложение, что, в свою очередь, позволит компилятору сделать дополнительные оптимизации. При этом размер кода будет увеличиваться.
- Java использует один и тот же код для всех настроек и тем самым минимизирует размер кода, но ухудшает производительность, так как во всех случаях будет работать один, неоптимизированный код.

¹¹ Под *декларативным* я (здесь) понимаю язык, которые описывает **что** делать, а не **как** делать.

¹² Как я уже писал в [1]: производительность не нужна только в том случае, если она есть. А чтобы она была, "лишнего" ухудшения производительности надо избегать.

Другие языки, например, С#, используют смешанную технологию, когда часть настроек делается через репликацию кода, а часть нет. Очевидно, что смешанный подход усложняет и так непростую реализацию.

Тривиль: Дженерики? Нет, обобщенные модули

Пока я не начал писать этот раздел статьи, я считал, что в Тривиле сделаны дженерики, необычным образом, но все же дженерики. Если подходить строго, дженерики подразумевают использование типового параметра, вместо которого могут быть подставлены разные типовые аргументы. В Тривиле сделано не так, поэтому строго говоря, термин дженерик использовать нельзя, хотя это, безусловно, обобщенные конструкции. Второе существенное отличие, это то, что единицей обобщения является модуль.

Начну с примера на Тривиле. Рассмотрим простейший контейнер: обобщенный стек (см. std/контейнеры/стек.tri [7]):

```
модуль стек

тип Элементы = []Элемент

тип Стек* = класс {
    элементы = Элементы[]
    верх: Цел64 := 0 // индекс свободной ячейки
}

фн (с: Стек) положить*(э: Элемент) {
    если с.верх = длина(с.элементы) {
        с.элементы.добавить(э)
    } иначе {
        с.элементы[с.верх] := э
    }
    с.верх++
}
// другие методы
```

Этот модуль выглядит обычным образом, без каких-либо дополнительных конструкций, за одним исключением — тип Элемент в нем не описан, он просто отсутствует. Сам по себе этот модуль недоопределен и не может быть использован, при попытке компиляции будет выдана ошибка.

Для того, чтобы его использовать, надо добавить то, что было не определено. Для этого есть специальная синтаксическая конструкция (см. std/контейнеры/стек/стек-стр.tri в [7]):

```
настройка "std::контейнеры/стек"
модуль стек-стр

тип Элемент = Строка
```

Настройка должна добавлять то, что отсутствует в исходном модуле. Настроенный модуль можно импортировать обычным образом и использовать:

```
модуль пример

импорт "std::контейнеры/стек/стек-стр"

вход {
    пусть с = стек-стр.Стек{}
    с.положить("Вася")
}
```

Конкретный или настроенный модуль собирается из двух частей. Сборка двух частей — это простое действие, и оно уже реализовано в компиляторе, так как любой модуль в Тривиле может состоять из нескольких исходных файлов. При компиляции любого модуля для каждого файла модуля строится АСД, в вершине которого узел для модуля, а потом верхние узлы сливаются в один. Для слияния двух узлов модуля надо соединить импорты и описания верхнего уровня.

После слияния следующие этапы компиляции, а именно семантическая проверка и генерация кода - выполняется над модулем целиком. В случае обобщенного модуля, если настройка неверна, например, задан неподходящий тип, то компилятор выдаст ошибку.

Вспомним теперь проблемы дженериков, которые мы рассматривали выше.

- сложности семантики.
 - Тривиль: сложности нет. Есть только конкретные типы, обработка которых не отличается от обычных типов, собственно, они и есть обычные типы;
- ограничения на типовые параметры (constraints).
 - Тривиль: отдельная синтаксическая конструкция для ограничений не нужна, собранный модуль или скомпилируется (настройка верная), или нет (настройка неверная).
- баланс между производительностью и размером кода.
 - Тривиль: проблемы нет, выбор сделан в сторону производительности. Теоретически проблема с размером кода может быть (как и в C++), но для Тривилиа это несущественно.

Как видим, этот подход, вместо решения проблем, их выкидывает: нет проблемы, не надо решать.

Трудоемкость решения минимальна, в компиляторе пришлось написать 120 строк (сто двадцать строк кода) для реализации обобщенных модулей, при том, что реализация дженериков в более-менее полном виде потребовала бы написать несколько тысяч строк кода.

Любопытно, что есть еще, по крайней мере, один язык, в котором используется обобщение на уровне модулей. Это язык СЗ [8], о котором я узнал уже после того, как язык и компилятор Тривилиа были сделаны. К сожалению, я не нашел в документации по этому языку обоснования использования обобщенных модулей.

Неожиданным следствием подхода Тривилиа является то, что модуль может быть настроен не только типом, но и константой, функцией и даже методом. Например, для настройки контейнера Словарь надо указать два типа (ключ, значение) и хеш-функцию.

Возникает вопрос: действительно ли этот подход является хорошим? Не совсем, он часто неудобен в использовании (необходимо писать отдельную настройку), есть и другие недостатки. Подход скорее остроумный, чем хороший. Впрочем, он позволил с минимальными усилиями реализовать необходимый контейнер и дал возможность идти дальше.

Я вполне допускаю, что его можно довести до хорошего, но тут надо думать.

Недостатки или точки роста

В статье рассмотрены языковые конструкции, для которых удалось найти простые, ограниченные, но достаточные решения, что позволило минимизировать трудозатраты. В следующей статье будет рассмотрен график разработки, и результат этих усилий будет наглядно виден.

Замечу еще, что внесенные ограничения в языковые конструкции можно считать недостатком языка. С другой стороны, любое ограничение создает **точку роста**. Все что сделано в Тривиле хорошо, может быть использовано в следующих языках как есть (впрочем, это не значит, что обязательно будет использовано как есть), а вот ограничения, такие как отсутствие итераторов, ограничение инициализации объектов, недостатки обобщенных модулей, — это то, на чем стоит сосредоточить внимание и улучшить в следующих языках.

Следующая статья будет посвящена реализации языка. В ней будет рассмотрена и обоснована архитектура компилятора и, как уже было сказано, график разработки и тоже с обоснованиями..

Литература

1. Недоря А. Е. Разработка языка Тривиль. Первые шаги к семейству языков. Часть 1, 2024. <http://digital-economy.ru/stati/разработка-языка-тривиль-первые-шаги-к-семейству-языков-часть-1>
2. Недоря А. Е. Разработка языка Тривиль. Часть 2, 2024. <http://digital-economy.ru/stati/разработка-языка-тривиль-часть-2>
3. The Swift Programming Language. Two-Phase-Initialization. <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/initialization/#Two-Phase-Initialization>
4. Kogtenkov A.V. Null safety benchmarks for object initialization. Proceedings of the Institute for System Programming of the RAS (Proceedings of ISP RAS). 2017;29(6):135-150. [https://doi.org/10.15514/IS-PRAS-2017-29\(6\)-7](https://doi.org/10.15514/IS-PRAS-2017-29(6)-7)
5. Language Mechanics On Escape Analysis, <https://www.ardanlabs.com/blog/2017/05/language-mechanics-on-escape-analysis.html>
6. T. Hume, Models of Generics and Metaprogramming: Go, Rust, Swift, D and More. <https://thume.ca/2019/07/14/a-tour-of-metaprogramming-models-for-generics/>
7. Тривиль: публичный репозиторий. <https://gitflic.ru/project/alekseinedoria/trivil-0>

8. C3. Generics. <https://c3-lang.org/generic-programming/generics/>

Спецификации языков программирования

- [Eiffel] ECMA International: Standard ECMA-367 – Eiffel: Analysis, Design and Programming Language 2nd edition (June 2006). <https://ecma-international.org/publications-and-standards/standards/ecma-367>
- [Go] The Go Programming Language Specification. <https://go.dev/ref/spec>
- [Kotlin] Kotlin language specification. <https://kotlinlang.org/spec/kotlin-spec.html>
- [Oberon] The Programming Language Oberon. <https://people.inf.ethz.ch/wirth/Oberon/Oberon.Report.pdf>
- [Rust] The Rust Reference. <https://doc.rust-lang.org/reference/>
- [Swift] The Swift Programming Language. <https://docs.swift.org/swift-book/documentation/the-swift-programming-language/>

References in Cyrillics

1. Недоря А. Е. Разработка языка Trivil`. Pervy`e shagi k semejstvu yazy`kov. Chast` 1, 2024. <http://digital-economy.ru/stati/razrabotka-yazyka-trivil`-pervye-shagi-k-semejstvu-yazykov-chast`-1>
2. Недоря А. Е. Разработка языка Trivil`. Chast` 2, 2024. <http://digital-economy.ru/stati/razrabotka-yazyka-trivil`-chast`-2>
3. Trivil: publicny`j repozitorij. <https://gitflic.ru/project/alekseinedoria/trivil-0>

Ключевые слова

язык программирования, семейство языков программирования, разработка языков программирования, компилятор, прототипирование компиляторов, энергосбережение разработчика

*Недоря Алексей Евгеньевич, к.ф.-м.н.
ORCID 0000-0001-8998-7072
aleksei.nedoria@yandex.ru
Телеграмм канал: t.me/vorchalki_o_prog*

Aleksei Nedoria, Development of programming language Trivil. Part 3. Balance.

Keywords

programming language, family of programming languages, programming language development, compiler, compiler prototyping, developer energy saving

DOI: 10.34706/DE-2025-01-03

JELclassification

Abstract

The article is the third in a series of articles that describe the development of the Trivil programming language. In previous articles, the goals of the language, the requirements for the language were defined and the main language solutions were described. This article examines complex language constructs, the development of which focused on the balance between the completeness of the construct and the complexity of the language and compiler. Like the entire series, the article is aimed, for the most part, not at the programmer who uses the language, but at the developer of programming languages.

УДК: 004.43, 004.4'4

1.4. Разработка языка Тривиль. Часть 4. Реализация

А. Е. Недоря, г. Санкт-Петербург

Статья является заключительной в серии статей, описывающих разработку языка программирования Тривиль и его реализацию. В предыдущих статьях речь шла о разработке собственно языка. Эта статья посвящена реализации языка. В ней рассматривается архитектура компиляторов, ход разработки и влияние решений на трудоемкость и скорость разработки. В статье также описывается область применимости языка.

Введение

В первых трех статьях серии были описаны, с некоторой степенью детализации, язык Тривиль. Основное внимание уделялось тому, как из целей и требований к языку, определенных в [1] вывелись конструкции языка и ограничения.

Основная идея, которая привела к разработке Тривилья — это переход к интенсивному программированию [4], путь к достижению которого я вижу через разработку семейства языков программирования и технологии архитектурного программирования.

Впрочем, интенсивное программирование тоже не самоцель, а средство для создания торгово-промышленно-финансового интернета и в конечном счете, для экспансии человечества в космос. Создание Тривилья — это первый шаг в этом направлении.

Следующий шаг уже начат, но еще не завершен. В следующих статьях я начну описывать то, что уже понято и сделано. Впрочем, Тривиль оказался полезным сам по себе и принес неожиданные результаты, об этом поговорим в завершающей части статьи.

Разработка языка программирования не имеет смысла без его реализации, а именно, без создания инструментария, позволяющего выполнить программу, написанную на этом языке.

В минимальный инструментарий входит компилятор, библиотека и среда исполнения. Без такого минимума язык является, в лучшем случае, набором идей или математической абстракцией.

Для широко используемого языка инструментарий должен быть существенно шире минимального, упомяну интеграцию со средами разработки (IDE), системы сборки, возможность подключения кода на других языках (interoperability), тесты на соответствие компилятора языку (compliance test suite) и так далее. Кроме того, нужна документация, включая спецификацию языка, описание библиотек, руководства разного уровня, а также презентационные материалы, в состав которых должен входить онлайн-компилятор (playground) и многое другое.

Требования к инструментарию и документации приводят к тому, что для разработки промышленного языка необходимо затратить сотни человеко-лет разработчиков разных специализаций (специалисты по языкам, компиляторам, библиотекам, тирингованию, интеграции, документации, внедрению и т.п.). Для примера укажу время разработки нескольких современных языков, начиная от объявленного разработчиками языка времени начала разработки до выхода первой (1.0) версии языка и инструментария:

- Go: 2007 - 2012 (около 5 лет),
- Swift: 2010 - 2014 (около 4 лет),
- Kotlin: 2010 - 2016 (около 6 лет).

При взгляде на эти цифры может показаться, что разработка языка малыми усилиями и в короткое время невозможна. Это действительно так, если речь идет о промышленном языке, в разработке которого всегда приходится учитывать множество конфликтующих требований (в том числе совместимости со старым кодом и другими языками), приходящих от разных заинтересованных сторон.

Ситуация существенно меняется, если речь идет об исследовательской работе, в рамках которой можно сконцентрироваться на главном, отбросить второстепенное и приложить максимальные усилия к упрощению языка и инструментов.

Разработка языка Тривиль началась 20 ноября 2022, а 9 сентября 2023, через 8.5 месяцев, был завершен принципиальный этап: язык, компилятор (точнее два компилятора) и среда исполнения были готовы к использованию на Windows и Linux. Собственно, точкой было завершение спецификации языка, остальное было сделано немного раньше.

В сентябре же 2023-го студенты Университета Иннополиса под руководством проф. Е.А. Зуева начали использовать язык на курсе разработки компиляторов, подробнее об этом позже.

Надеюсь, что у читателя возник вопрос: как и за счет чего можно ускорить реализацию и вместо сотен человеко-лет затратить несколько человеко-месяцев (хотя, скорее, надо считать в человеко-днях или часах).

Первая часть ответа на вопрос, почему Тривиль (язык и реализация) был сделан быстро и малыми усилиями, очевидна: у меня есть опыт подобных разработок, и я точно понимал, что и как надо сделать. Последовательность шагов разработки была для меня очевидна с самого начала:

1. набросок грамматики языка,
2. первый компилятор Тривилья, одновременно:

- разработка среды исполнения и первых библиотек,
- доработка синтаксиса и семантики языка;
- 3. черновик описания языка,
- 4. компилятор Тривилиа на Тривиле (bootstrap), одновременно:
 - доработка среды исполнения и библиотек,
 - доработка языка, включая учет потребности библиотек,
 - доработка компилятора на Go;
- 5. описание языка.

Вторая часть ответа тоже проста: я осознавал свою ограниченность, или, говоря словами Эдсгера Дейкстры: осознавал, что я “смиранный” программист. Моей постоянной мыслью была мысль: как сделать проще, как пройти коротким путем, но при этом достичь цели.

На мой взгляд, практически вся информатика (computer science) строится на принципиально неверном философском подходе, который хорошо выразил Гради Буч [5]: “...*сложность, о которой мы говорим, является, по-видимому, необходимым свойством всех больших программных систем. Под необходимостью мы имеем в виду следующее: можно создать эту сложность, но нельзя придумать так, чтобы обойтись без нее.*”

Другими словами: “*что тут думать, трясти надо!*”. И это проявление лени ума: давайте не будем решать задачу, потому что мы не знаем, как ее решить. Об этом стоило бы поговорить подробнее, но это, скорее, тема для следующих статей. Пока же напомним о другой школе: *make it as simple, as possible, but not simpler*¹. И если мы прямо сейчас не знаем и не умеем делать проще, то надо сделать, как можем и думать дальше. И еще: *Мы сами знаем, что она [задача] не имеет решения [...] Мы хотим знать, как её решать*². Подробнее см. [6].

Собственно, три первые статьи описывают то, как сделать простой, но достаточный язык. Тот же самый подход был применен к реализации компиляторов, библиотек и среды исполнения.

Два компилятора

В плане разработке (см. выше) указано, что надо разработать два компилятора. Первый, очевидно, надо писать не на Тривиле, так как Тривиль–компилятора еще нет.

Естественный вопрос: зачем писать второй компилятор? Я не буду отвечать на этот вопрос в общем, раскрутка компилятора (bootstrapping) давно применяется и полезность ее хорошо описана. Что же касается Тривилиа, то разработка второго компилятора является необходимой, так как язык позиционируется как язык для разработки компилятора. Компилятор Тривилиа на Тривиле является обязательной проверкой пригодности языка для целевой области.

Остается выбрать язык для первого компилятора. Этот выбор для меня был простым. Я уже писал компиляторы на Go, к тому же устройство программы на Тривиле похоже на устройство Go-программы. Это позволило использовать одинаковую архитектуру для обоих компиляторов.

Кроме того, несмотря на существенную идеологическую разницу языков, например, в подходе к безопасности ссылок и поддержки ООП, я понимал, как писать текст на подмножестве Go, так чтобы он был близок к тексту на Тривиле. Близость исходных текстов позволила на этапе разработки второго компилятора написать простую программу преобразования текста Go ⇒ Тривиль и существенно сократить трудоемкость.

Уточнение задачи компилятора

В общем виде задачу компилятора можно сформулировать так: по набору исходных текстов построить образ программы, готовый к исполнению.

Например, запуская `clang hello.c` на корректном исходном файле, мы получим исполняемую программу. Но если `hello.c` использует функцию, определенную в другом исходном файле, например, `aux.c`, то запуск уже должен быть `clang hello.c aux.c`, иначе используемая функция не будет найдена. Дело тут в том, что в языках семейства C/C++ устройство программы не определено на уровне языка и задается внешним образом, например, через перечисление исходных файлов в командной строке или в `makefile`.

В тех же языках, в которых устройство программы задается на уровне языка, например, в Go, для компиляции программы, состоящей из одного файла (пакета, модуля) или множества пакетов, достаточно указать головной модуль программы, например, команда `go build tric.go` компилирует целиком компилятор Тривилиа, написанный на Go.

Тривиль, как и Go — это модульный язык с заданным (через импорт) устройством программы, что позволяет компилировать программу целиком.

Теперь мы можем уточнить задачу компилятора при компиляции программы, разбив ее на подзадачи:

- собрать список всех модулей, составляющих программу,
- скомпилировать каждый из этих модулей (в правильном порядке),
- собрать исполняемую программу.

Разбиение на подзадачи почти позволяет перейти к архитектуре компилятора, но надо уточнить еще один вопрос, существенно влияющий на сложность компилятора.

¹ Цитата А.Эйнштейна, использованная Никлаусом Виртом как лого языка Oberon.

² Цитата из “Понедельник начинается в субботу”, А. и Б. Стругацкие

Раздельная компиляция

Под **раздельной компиляцией** понимается возможность использовать при компиляции программы результат предыдущих компиляций частей программы (модулей или исходных файлов). Раздельная компиляция является обязательной для работы с большими программными системами, так как компиляция всех исходных текстов может занимать существенное время. Раздельная компиляция позволяет при повторной компиляции обрабатывать только те модули, которые изменились (прямо или косвенно), и существенно ускоряет процесс компиляции.

Для языков без импорта, таких как C/C++, раздельная компиляция не требует никакой поддержки компилятора, проверка совместимости сущностей программы происходит при сборке (линковке). Такой вариант раздельной компиляции обычно называется **независимой компиляцией**.

Для модульных языков поддержка раздельной компиляции является непростой задачей и требует проверки совместимости модулей. Например, если модуль А импортирует Б и модуль Б был изменен, то в общем случае нельзя использовать результат предыдущей компиляции как модуля Б, так и А, их надо компилировать заново.

Алгоритм определения минимального набора модулей, которые надо компилировать, является непростым, при этом его сложность существенно зависит от требований. Например, пусть в модуль Б была добавлена функция, которую модуль А не использует. Можно ли при этом использовать результат предыдущей компиляции А? Это далеко не простой и не единственный возникающий вопрос.

Но даже если использовать самый простой способ определения минимального набора модулей, компилятор, поддерживающий раздельную компиляцию, должен уметь записывать информацию об экспортированных сущностях модуля и читать её быстро. Иначе никакой выгоды от раздельной компиляции не будет. Задача сериализация/десериализация сложной структуры данных весьма нетривиальна.

Итак, поддержка раздельной компиляции в компиляторе требует существенных усилий, но является ли она необходимой для Тривиля?

На мой взгляд, нет. Компилятор Тривиля не является большой программной системой, и полная компиляция всех его модулей должна происходить быстро (и это подтверждено практикой).

Отказ от раздельной компиляции – это еще один пример упрощения, только теперь не языка (см. [3]), а компилятора.

Замечу, что для следующих языков семейства раздельная компиляция является необходимой и обязательно будет сделана, но сначала до этих не-Тривиль языков надо добраться.

Архитектура компилятора

После уточнения задачи компилятора можно перейти к рассмотрению архитектуры компилятора. На верхнем уровне компилятор состоит из двух логических частей, первая обрабатывает программу целиком, вторая работает с модулем.

Компилятор всегда вызывается на головном модуле программы и работает схематически так:

1. выполняет синтаксический анализ головного модуля и рекурсивно всех модулей, которые импортирует каждый обработанный модуль;
2. строит упорядоченный список модулей, начиная с тех модулей, которые ничего не импортируют и заканчивая головным модулем;
3. для каждого модуля из этого списка выполняет семантический анализ и генерацию;
4. собирает исполняемую программу.

Как видно из этого описания, сначала выполняется синтаксический анализ для всех модулей программы, при этом лексический анализ является составной частью синтаксического. После завершения синтаксического анализа программа представлена в виде линейного списка модулей, где каждый модуль представлен своим абстрактным синтаксическим деревом (АСД). При этом, если исходно модуль состоит из нескольких файлов, то на этом уровне для каждого модуля уже выполнено объединение АСД-файлов в АСД-модуля. Во время объединения также выполняется настройка обобщенных модулей, и далее на стадии семантического анализа и генерации обобщенных модулей не существует, что существенно упрощает компилятор.

Для АСД каждого модуля выполняется последовательно несколько проходов семантического анализа и проход генерации выходного языка.

Семантический анализ

Основная задача семантического анализа — проверка корректности конструкций языка, которая разбивается на две подзадачи, каждая из которых решается на отдельном проходе:

- разрешение имен и областей видимости,
- контроль типов.

Семантический анализ получает на вход АСД после синтаксического анализа и дорабатывает его.

Результатом первого прохода для корректного модуля является АСД, в котором для каждого использования идентификатора указан объект, который идентификатор обозначает в точке использования, а это может быть:

- предопределенный объект, например, тип Строка или константа истина,
- объект (тип, константа или функция), описанный в данном модуле в одной из объемлющих точку использования областей видимости,

- или импортированный объект.

На втором проходе проверяется совместимости типов для всех операций языка, кроме того, в случае некоторых неявных преобразований АСД достраивается, чтобы преобразование стало явным и было представлено узлом в АСД.

Разбиение семантического анализа на два последовательных прохода позволяет существенно упростить компилятор. Во-первых, за счет того, что решение одной задачи проще, чем решение двух (single responsibility principle), во-вторых, контроль типов всегда получает на вход АСД, в котором нет ошибок, связанных с именами.

Архитектурно семантический анализ сделан так, чтобы в него легко можно было добавить дополнительные проходы, например, оптимизирующие проходы.

Генерация

Задача генерации — отобразить АСД на выходной язык, который, в общем, может быть кодом процессора, байт-кодом виртуальной машины, промежуточным языком или языком программирования.

С точки зрения минимизации усилий генерация кода конкретного процессора неприемлема, так как ограничивает переносимость компилятора и программ. Генерация байт-кода также не лучший вариант, так как добавляет существенную зависимость от виртуальной машины.

По сути, два варианта, которые имеет смысл рассматривать — это генерация LLVM IR или текста на языке C. Оба этих варианта обеспечивают:

- переносимость,
- качество кода, за счет оптимизаций на уровне LLVM или C,
- относительную простоту генерации.

При этом генерация кода на C проще, так как уровень языка выше и не нужно учитывать специфику LLVM IR, а учитывать её в некоторых местах довольно сложно, как показывает мой опыт генерации LLVM IR.

Замечу, что я не рассматриваю генерацию на другие языки программирования, язык C — это единственный из распространенных языков, который является переносимым ассемблером и занимает промежуточное место между языками высокого и низкого уровня. Писать программы на C не полезно, но при использовании в качестве промежуточного языка многие его недостатки как языка программирования становятся достоинствами. Например, отсутствие проверки индекса массива является недостатком для языка программирования, но достоинством для промежуточного языка, так как позволяет корректно отобразить в коде семантику исходного языка.

По совокупности генерация кода на C существенно выигрывает в простоте реализации и удобству отладки по сравнению со всеми другими вариантами. Для генерации используется стандарт C99, а для получения исполняемой программы, обычно, компилятор clang.

Устройство компилятора

Теперь можно перечислить модули, из которых состоит компилятор³:

- трик: головной модуль, обработка командной строки, запуск компиляции или других действий;
- асд: описание классов абстрактного синтаксического дерева;
- компилятор: компиляция программы,
- парсер: синтаксический анализ,
 - лексер: лексический анализ;
- семантика,
 - имена: разрешение имен,
 - контроль: контроль типов;
- генерация,
- основа: обеспечение работы, включая чтение/запись файлов и обработку ошибок.

Устройство простое и, на мой взгляд, идеальное, полученное исходя из опыта работы над предыдущими компиляторами. Основное здесь — это последовательное применение принципа единственной ответственности (single responsibility principle). При этом модули достаточно крупные, дробление их на более мелкие привело бы к увеличению количества связей и увеличению сложности программирования и сопровождения.

Компилятор и ООП

Если спросить меня, в какой парадигме написан компилятор, то я скажу — в ООП. Но если приглядеться, то это весьма странный ООП, который кто-то вообще не признает за ООП.

Рассмотрим АСД. Оно состоит из объектов классов, и все классы являются наследниками класса Узел. Вот сокращенная диаграмма классов, корень которого класс Узел:

Узел

- Тип
 - ТипВектор
 - ТипКласс
 - *другие типы*

³ Как уже упоминалось, оба компилятора, компилятор на Go и компилятор на Тривиле, архитектурно одинаковы.

- Описание
 - ОписаниеКонстанты
 - ОписаниеФункции
 - *другие описания*
- Оператор
 - ОператорПрисвоить
 - ОператорЕсли
 - *другие операторы*
- Выражение
 - *разные выражения*

ООП как есть. Но вот в чем вопрос: а что с **инкапсуляцией**? На каком уровне нужна инкапсуляция? Ответ: *каждый класс должен быть защищен от других* - не подходит. Любой из этих классов отдельно не имеет смысла. АСД — это одна структура данных, состоящая из объектов многих классов, ставить защиту между ними так же бессмысленно, как между полями одного класса.

Тогда может быть нам нужна *инкапсуляция на уровне модуля*? Опять мимо, АСД — это структура, которая обрабатывается всеми проходами компилятора. Естественно сравнить АСД с заготовкой, которую двигают по конвейеру и последовательно обрабатывают разными станками. Если станок не должен изменять какие-то части заготовки, то это настройка станка, а не заготовки. К тому же, на конвейер можно поставить дополнительный станок (проход компилятора). Неужели из-за этого надо менять заготовку? Это задача станка, и это снова принцип единственной ответственности.

Может быть, третий ответ: если у класса есть метод, для его работы нужны дополнительные данные, то доступ к ним имеет смысл контролировать. Вроде бы разумно, но давайте посмотрим, какие это могут быть методы.

Мы знаем, что узел АСД последовательно обрабатывается проходами, например, семантическими. Кажется правильным добавить, например, метод проверить тип для каждого узла выражения. Это же здорово, проход контроля типов просто вызывает этот метод на каждом узле и готово.

Но что мы тогда получим:

- мы переместили весь сущностный код проверки типов в описание классов и **потеряли инкапсуляцию** на уровне прохода: теперь все проходы имеют доступ к проверке типа;
- каждый класс имеет собственный метод, хотя для многих операций, например, для бинарных операций "+" и "-" проверка одинаковая, и это приводит нас к дублированию кода или к необходимости хитрить, чтобы не дублировать;
- если для проверки типа нам надо выйти за пределы одного узла, то нам придется снова хитрить.

Но это были мелкие неудобства. Если мы добавили метод (или методы) для проверки типа, то естественно добавить методы для других семантических проверок, а также для генерации кода и для других дополнительных действий. Например, Тривиль позволяет записать АСД в виде списочного выражения (S-expression), чтобы его можно было удобно визуализировать. А еще, выдать экспортированные сущности модуля в виде, удобном для чтения. Оба этих действия тоже работают с АСД. Если мы будем последовательны, нам придется добавить десятки методов в каждый класс. Это приведет к тому, что практически весь код компилятора окажется в модуле асд, и мы получим то, что называется *спагетти-код*.

Как видно, неверный подход и ложно понятая инкапсуляция приводит к программному кошмару и к полной потере инкапсуляции. А какая инкапсуляция нам нужна?

Если задуматься, ответ очевиден: нам нужна инкапсуляция на уровне прохода (обрабатывающего станка). Каждый проход решает свою задачу, использует свои алгоритмы для работы, и ему нужны свои локальные данные. Никакие **локальные данные прохода не должны попасть в АСД**, даже если это кажется удобным и увеличивающим скорость работы. Джендльмен в контроле типов не отвечает за то, что делает джендльмен в генерации. И не только не отвечает, но и ничего не знает об этом.

Такая изоляция позволяет менять код прохода независимо от других проходов, добавлять дополнительные проходы, менять одну генерацию на другую, то есть превращает компилятор в конструктор (или конвейер) из компонент.

Какие же тогда методы есть смысл делать у классов, описывающих узлы АСД? Наверно те, которые не связаны с обработкой, а отображают некоторые свойства объектов. Сколько таких объектов есть в АСД Тривилиа? Ответ: **ни одного**.

В модуле асд описаны 64 класса, и ни у одного из них нет методов. Замечу, что если бы мне понадобился какой-то полезный метод, я бы его добавил, но ни разу не понадобилось.

Соответственно, вторая священная корова ООП — **полиморфизм**, в данном месте «сдохла». Нет методов, нет полиморфизма. А вот **наследование** используется как инструмент структурирования и дополнительного контроля типов, позволяющий разделить разные сущности, такие как описания, операторы и выражения.

Еще раз, методы нужны, но там, где они уместны. Например, каждый проход компилятора определяет объект, содержащий локальные данные прохода и методы, обрабатывающие АСД и отдельные узлы. Здесь методы есть, инкапсуляция есть (на уровне модуля), а наследования нет, и полиморфизма тоже нет.

На мой взгляд, обычная реализация полиморфизма, а именно: возможность переопределить метод и использовать метод базового типа (супертипа) — это в 99% случаев архитектурная ошибка⁴. Правильное использование полиморфизма — это реализация абстрактного метода. Допустимое использование — это новая реализация метода в подтипе, не зависящая от реализации в супертипе.

Тривиль разрешает переопределение метода, но не позволяет вызывать метод супертипа. На мой взгляд, это правильный ООП, который надо дополнить средствами утиной типизации, но это другая тема.

Ход и итоги разработки

В этой и в предыдущих статьях много раз говорилось об упрощении языка и компилятора. Остается показать, как эти упрощения сказались на ходе разработки. Длительность этапов в часах рассчитывается по средней нагрузке 8 часов за неделю (приблизительный расчет).

Дата	Событие	Этап и длительность
20.11.2022	Начало разработки	<ul style="list-style-type: none"> ● Цели и требования ● Черновая грамматика 2 недели/16 часов
02.12.2022	Выложен первый исходный текст компилятора на Go (первый коммит)	<ul style="list-style-type: none"> ● Разработка компилятора на Go ● Разработка среды исполнения ● Разработка первых библиотек 9 недель/72 часа
03.02.2023	Компилятор Тривили на Go компилирует все основные конструкции, код тестов выполняется.	
19.02.2023	Начало написания спецификации языка (черновик)	<ul style="list-style-type: none"> ● Разработка черновой спецификации ● Доработка компилятора 3 недели/24 часа
16.03.2023	Выложен первый исходный текст компилятора на Тривиле (первый коммит)	<ul style="list-style-type: none"> ● Разработка компилятора на Тривиле ● Доработка компилятора на Go ● Разработка библиотек, необходимых компилятору 14 недель/112 часов
25.06.2023	Компилятор на Тривиле компилирует сам себя. Достигнута неподвижная точка (см. ниже)	
07.09.2023	Спецификация языка версии 0.9 завершена	<ul style="list-style-type: none"> ● Доработка спецификации, компиляторов, среды исполнения и библиотек ● Перенос на другие платформы⁵ 10 недель/80 часов

Поясню, что понимается в таблице под **неподвижной точкой**. Применяем компилятор на Go к исходным текстам компилятора на Тривиле, получаем набор исходных текстов на языке Си. Из этих исходных текстов собираем исполняемый код компилятора (используем clang). Новым компилятором компилируем те же исходные тексты, получаем второй набор исходных текстов на языке Си. Сравниваем эти исходные тексты. Если они полностью совпадают, то это и есть неподвижная точка. Сколько бы раз мы ни компилировали вновь собранным компилятором его же исходные тексты, результат не изменится.

Достижение неподвижной точки является существенным признаком работоспособности и качества компилятора, так как компилятор для самого себя является достаточно большим и сложным тестом.

⁴ а в оставшихся случаях, это грубая архитектурная ошибка.

⁵ Перенос на Linux, FreeBSD, MacOS выполнен Дмитрием Соломенниковым и Иваном Тюляндиным

Замечу, что после достижения неподвижной точки, разработка компилятора на Go была практически заморожена (кроме исправления ошибок), а доработка и улучшения компилятора на Тривиле продолжилась. Так что сейчас неподвижной точки уже нет.

Приведу размеры исходных текстов на начало сентября 2023 года:

Программная часть	Язык	Размер в строках
Компилятор на Go	Go	11,200
Компилятор на Тривиле	Тривиль	10,500
Среда исполнения	C99	1,800
Библиотеки	Тривиль	2.300

Итого, меньше чем за 9 календарных месяцев, минимальными усилиями (8 часов в неделю), написаны два компилятора, среда исполнения и библиотеки, суммарно 25 тысяч строк на трех языках программирования [6].

На мой взгляд, это существенное подтверждение мысли, что упрощение работает и делает невозможные вещи не просто возможными, а обозримыми и выполнимыми.

Понятно, что для широкого или коммерческого использования надо еще много сделать, но то, что уже сделано — это не MVP (minimum value product), это не прототип на выброс, а рабочий продукт, который активно используется для разработки следующих языков.

Области применения

Основная область применения Тривили очевидна и совпадает с заявленной целью языка: разработка компиляторов. Язык и компилятор используются для разработки следующих языков семейства, языков Арс и Арвиль. О них я буду писать в следующих статьях.

Вторая область стала видна уже в ходе разработки, это **полигон для обучения** студентов разработке языков, компиляторов, инструментов (например, статических анализаторов), библиотек и т.д. Простота языка и компилятора позволяет в рамках студенческого курса сделать практическую работу по множеству направлений, подробнее см. [8].

На практике использование Тривили было проверено в 2023-2024 учебном году в Университете Иннополиса. В рамках курса по разработке компиляторов под руководством проф. Е.А. Зуева группы студентов делали генерацию кода для Тривили в LLVM IR, JVM и .NET, и еще одна группа сделала набор тестов на соответствие компилятора спецификации языка. Все группы успешно выполнили учебную задачу, и хотя ни одна из групп не добилась включения результатов работы в основной репозиторий Тривили, но такая задача перед ними не ставилась.

Заключение

Разработка Тривили была (и есть) большим экспериментом, попыткой “впихнуть невпихуемое” и прорваться к звездам. Эксперимент продолжается, и о его развитии будут следующие статьи.

Моя огромная благодарность всем, кто участвовал, критиковал, предлагал решения и поддерживал меня на этом пути. Отдельная благодарность: Е. Зуев, А. Канатов, Д. Соломенников, И. Туляндин, А. Чесноков, Н. Шилов. А также всем участникам семинара STEP⁶ [9], Университету Иннополиса и журналу “Цифровая экономика”.

Тривиль — это личинка хорошего языка программирования, и из него, при вложении сил, может вылупиться бабочка. Но мы пойдем другим путем.

Литература

1. Недоря А. Е. Разработка языка Тривиль. Первые шаги к семейству языков. Часть 1, 2024. <http://digital-economy.ru/stati/разработка-языка-тривиль-первые-шаги-к-семейству-языков-часть-1>
2. Недоря А. Е. Разработка языка Тривиль. Часть 2, 2024. <http://digital-economy.ru/stati/разработка-языка-тривиль-часть-2>
3. Недоря А. Е. Разработка языка Тривиль. Часть 3. Баланс, 2024. <http://digital-economy.ru/stati/разработка-языка-тривиль-часть-3-баланс>
4. Недоря А. Е. Интенсивное программирование, 2022. <http://digital-economy.ru/stati/интенсивное-программирование>
5. Буч Г. Объектно-ориентированное проектирование с примерами применения, М.: Конкорд, 1992.

⁶ Семинар по фундаментальным вопросам программной инженерии, теории и экспериментальному программированию (Software Engineering, Theory and Experimental Programming)..

6. Непейвода, Н. Н. Методы Кристобая Хунты, https://ru.wikibooks.org/wiki/Методы_Кристобая_Хунты
7. Тривиль: публичный репозиторий. <https://gitflic.ru/project/alekseinedoria/trivil-0>
8. Нодоря А. Е. Язык программирования для обучения технологиям компиляции и трансформации. Труды Института системного программирования РАН. 2023;35(6):95-102. [https://doi.org/10.15514/ISPRAS-2023-35\(6\)-5](https://doi.org/10.15514/ISPRAS-2023-35(6)-5)
9. Семинар STEP, <https://persons.iis.nsk.su/en/STEP-2023>, <https://persons.iis.nsk.su/en/STEP-2024>

Спецификации языков программирования

[Go] The Go Programming Language Specification. <https://go.dev/ref/spec>

[Kotlin] Kotlin language specification. <https://kotlinlang.org/spec/kotlin-spec.html>

[Oberon] The Programming Language Oberon. <https://people.inf.ethz.ch/wirth/Oberon/Oberon.Report.pdf>

[Rust] The Rust Reference. <https://doc.rust-lang.org/reference/>

[Swift] The Swift Programming Language.

<https://docs.swift.org/swift-book/documentation/the-swift-programming-language/>

References in Cyrillics

1. Nedorya A. E. Razrabotka yazy`ka Trivil`. Pervy`e shagi k semejstvu yazy`kov. Chast` 1, 2024. <http://digital-economy.ru/stati/razrabotka-yazyka-trivil-0-pervye-shagi-k-semejstvu-yazykov-chast-1>
2. Nedorya A. E. Razrabotka yazy`ka Trivil`. Chast` 2, 2024. <http://digital-economy.ru/stati/razrabotka-yazyka-trivil-0-chast-2>
3. Nedorya A. E. Razrabotka yazyka Trivil`. Chast' 3. Balans, 2024. <http://digital-economy.ru/stati/razrabotka-yazyka-trivil-0-chast-3-balans>
4. Nedorya A. E. Intensivnoe programmirovaniye, 2022. <http://digital-economy.ru/stati/intensivnoe-programmirovaniye>
5. Buch G. Ob`ektno-orientirovannoye proektirovaniye s primerami primeneniya, M.: Konkord, 1992.
6. Trivil: publichny`j repozitorij. <https://gitflic.ru/project/alekseinedoria/trivil-0>
7. Nepejvoda, N. N. Metody Kristobalya Hunty, https://ru.wikibooks.org/wiki/Методы_Кристобая_Хунты
8. Nedorya A. E. Yazyk programmirovaniya dlya obucheniya tekhnologiyam kompilyacii i transformacii. Trudy Instituta sistemnogo programmirovaniya RAN. 2023;35(6):95-102.

Нодоря Алексей Евгеньевич, к.ф.-м.н.

ORCID 0000-0001-8998-7072

aleksei.nedoria@yandex.ru

Телеграмм канал: t.me/vorchalki_o_prog

Ключевые слова:

язык программирования, семейство языков программирования, разработка языков программирования, типовая система, компилятор, архитектура компиляторов.

Aleksei Nedoria, Development of programming language Trivil. Part 4. Implementation.

Keywords

programming language, family of programming languages, programming language development, type system, compiler, compiler architecture, developer energy saving.

DOI: 10.34706/DE-2025-01-04

JEL classification:

Abstract

The article is the final in a series of articles describing the design of the Trivil programming language and its implementation. In previous articles, we talked about the development of the language itself. This article is about the implementation of the language. It examines the architecture of compilers, the course of development, and the impact of solutions on the complexity and speed of development. The article also describes the scope of the language.

УДК 004.9

1.5. Цифровая трансформация торгово-промышленных предприятий на основе использования инновационных практик

С.С. Никитчук¹, О.В. Китова.¹¹РЭУ им. Г.В. Плеханова. Москва, Россия

Введение. В статье приведен обзор организационной и процессной проблематики цифровой трансформации торгово-промышленных предприятий. Разработан подход к проведению качественной и эффективной цифровой трансформации и построению инновационной бизнес-модели.

Методы исследования. Основным методом исследования является системный анализ современных подходов в области трансформации организационной и процессной моделей предприятия, которые помогают крупным компаниям не только осуществить трансформацию, но и совершить прорыв (Disrupt) с помощью ИТ как ключевого драйвера изменений.

Результаты и дискуссия. Основным результатом работы является подход к проведению качественной и эффективной цифровой трансформации и построению инновационной бизнес-модели. Опыт авторов и многочисленные бизнес-кейсы иностранных (Apple, Netflix, Amazon, др.) и отечественных компаний (Сбер, Х5, Ингосстрах, др.) показывают, что компании находятся в постоянном поиске применения современных технологий, что требует быстрой реализации идей от инициации до реализации (MVP – начальная версия продукта) и до вывода продукта на рынок. Скорость является основным конкурентным преимуществом и ограничением. Каждый отдельный продукт может стать как прорывным, так и неудачным, поэтому необходимо быстро и недорого тестировать гипотезы на каждом этапе создания продукта (модель Stage-Gate).

При этом компании разделяют технологии, приносящие добавленную стоимость (операционные) и управленческие (регуляторные). Первые являются зачастую уникальными разработками (НИОКР), тогда как вторые – стандартами конкретного рынка/отрасли (ERP, CRM, др.). Они взаимосвязаны и интегрированы между собой. Поэтому предприятие должно инвестировать в первые и контролировать затраты на вторые.

Заключение. Важнейший вывод данной статьи заключается в том, что российские компании, которые занимаются проблемой обеспечения технологического суверенитета и хотят сохранить/получить лидерство на рынках, должны обратить внимание на современные подходы по построению эффективной модели цифрового предприятия перед тем, как внедрять те или иные ИТ-решения.

Введение

Цифровая трансформация сегодня играет ключевую роль в построении инновационного предприятия. В то время как торгово-промышленные предприятия сталкиваются с беспрецедентными внешними и внутренними вызовами, именно цифровая трансформация находит решения, которые помогают предприятиям выиграть в конкурентной борьбе. При этом нет и не может быть готового решения, так как в этом случае оно будет уже широко использовано на рынке и не принесет ожидаемых выгод. Разработка же индивидуальных решений зачастую стоит дорого и длится долго, что несет ряд финансовых и операционных рисков в современном мире. Поэтому компании ищут быстрые и качественные подходы для постоянной цифровой трансформации, своевременно внедряя новые решения и выводя из эксплуатации устаревшие (неэффективные).

Современный «хайп» (ажиотаж) на решения по искусственному интеллекту показывает не столько прорывную способность данной отдельной технологии, сколько возможность получения конкурентных преимуществ при синергии использования с текущими технологиями компании. Также решения по искусственному интеллекту выявили неготовность части предприятий к их промышленному использованию из-за применения устаревшей технологической базы, включая системы управления предприятием и корпоративные хранилища данных. Это привело к тому, что большое количество компаний планируют и внедряют крупные программы трансформации, включающие в себе десятки или сотни проектов в области организационных изменений, построения целевых бизнес-процессов и внедрения современных технологий.

Методы исследования

Большинство поставщиков технологических решений для бизнеса собирают статистику и агрегируют успешные кейсы достижения выгод. По опыту автора, адаптивность и современность предприятия подчеркивается ими как условие использования современных технологий. Предприятия должны быть «готовы» к цифровизации, сотрудники – иметь цифровые компетенции, инфраструктура – хранить и обрабатывать большие объемы данных для своевременного получения необходимой информации.

Данная проблематика формирует ключевую задачу использования современных технологий торгово-промышленного предприятия – построение организационно-функциональной модели и целевых процессов, которые позволят эффективно вести бизнес сегодня и быстро реагировать на изменения завтра. Ввиду частой изменчивости внешней среды данная модель и бизнес-процессы предприятия должны быть адаптированными к быстрым внедрениям преобразований и улучшений.

Основным методом исследования является системный анализ успешных кейсов цифровой трансформации и поиск закономерностей для построения фреймворка цифровой трансформации. Подходы к управлению программами трансформации определяют достижение предприятием ожидаемых выгод и стратегических целей.

Обзор литературы

Будущее предприятия определяется тем, насколько хорошо сотрудники организованы, процессы адаптируются и технологии совершенствуются. В работах авторов был дан анализ данной проблематики, приведен фреймворк цифровой трансформации [1] и подход к управлению кадрами для успешного проведения цифровой трансформации [2].

Обзор отечественной литературы и открытых источников информации показывает, что цифровая трансформация является основой стратегии компаний. Несмотря на виды деятельности и отрасль компаний, успешные кейсы базируются на применении современных информационных технологий и организационного развития на их основе. Ниже приведены примеры известных отечественных компаний:

- В период 2023-2026 Сбер планирует переход в человеко-центричную организацию, в основе которой три ключевых направления развития: технологии, бизнес-модель и культура [3].

- Яндекс позиционирует себя как диверсифицирующийся технологический гигант, основанный на сильной команде. Технологический фундамент Яндекса позволил стать одной из ведущих компаний в области развития искусственного интеллекта в мире [4].

- Маркетплейс ОЗОН стремится использовать передовые технологии, логистику и инновации, чтобы всегда быть на шаг впереди потребностей клиентов. В дополнение к основному бизнесу ОЗОН развивает сервисы, включая FinTech (Финансовые технологии) продукты, экспресс-доставку и бронирование путешествий [5].

- Росэнергоатом уделяет повышенное внимание собственной цифровой трансформации, которая включает следующие программы проектов: Цифровая энергетика, Цифровая АЭС, Цифровые продукты [6].

- Цифровая трансформация ОАК включает внедрение цифровых технологий в управлении жизненным циклом изделий, проектировании, испытаниях и сертификации, производстве, послепродажном обслуживании, управлении активами и персоналом [7].

Для реализации цифровой трансформации современные мировые лидеры в области программного обеспечения для предприятий (Microsoft, Oracle, SAP) [8] предлагают не просто набор технологий, а готовые кейсы автоматизации, так называемые лучшие практики цифровизации. Предприятие, внедряющее данные готовые кейсы, должно адаптировать свою бизнес-модель под процессы, содержащиеся в решении. Несмотря на высокую стоимость такого программного обеспечения, они предлагают проверенные на мировом рынке решения.

В качестве альтернативы крупным зарубежным системам управления предприятием на рынке РФ представлено достаточное количество отечественных решений (1С, Диасофт, Галактика, Турбо) [9], среди которых особо выделяются решения 1С как наиболее зрелые и универсальные [10]. Однако современные российские лидеры в области программного обеспечения для предприятий не имеют возможности внедрения в свои системы широкого набора бизнес-процессов (лучших практик), поэтому идут по пути создания гибких механизмов доработки своих решений, либо создания решений с использованием открытого кода [11]. Внедрения ИТ-решений выполняются совместно вендорами, заказчиками и отдельными разработчиками. Поэтому предприятия должны правильно формулировать бизнес-требования (feasibility study) для системных-аналитиков и технических специалистов, чтобы развитие системы («тюнинг») было эффективно с точки зрения стоимости и сроков достижения выгод.

Чтобы понять основные факторы успеха проектов, международный институт PMI опросил более 3.500 профессионалов. Исследование показывает, что организации, которые уделяют большое внимание управлению проектами и программами трансформации, значительно лучше справляются с достижением бизнес-целей. Они также характеризуются значительно меньшим числом отклонений в содержании и бюджете проектов [12].

Важной особенностью цифровой трансформации является её постоянство и этапность. Предприятие должно внедрять отдельные компоненты информационно-аналитической системы с учетом уровня ее цифровой зрелости, стратегических приоритетов и состояния развития цифровой экосистемы государства и современных мировых практик. В настоящее время не существует универсальной стратегии развития описанных цифровых моделей, поэтому каждое предприятие самостоятельно определяет траекторию собственного развития [13].

Обзор литературы и актуальных исследований показывает актуальность цифровой трансформации для современных предприятий и новизну исследования ввиду постоянного усложнения ИТ-решений и внешней среды предприятий.

Результаты и дискуссия

Референтные модели бизнес-процессов

Несмотря на то, что каждый бизнес уникален и формируется внешней и внутренней средой, предприятия стремятся унифицировать подходы и используемые технологии. Консалтинговые организации и бизнес-институты формируют референтные модели типовых бизнес-процессов, которые стандартизуют процессы, релевантные для всех или группы предприятий. Одним из самых известных таких фреймворков является классификатор типовых процессов (PCF, Process Classification Framework) от Американского центра производительности и качества APQC [14]. APQC – ведущая мировая организация в области бенчмаркинга, лучших практик, улучшения процессов и производительности, а также управления знаниями.

Основной моделью фреймворка является универсальный кросс-индустриальный классификатор, который содержит типовые процессы и применим к большинству предприятий. Данная модель расширяется на специализированные и индустриальные классификаторы, отражающие специфику следующих различных отраслей: Нефтегазовая, Финансовая, Металлургическая, Химическая, Фармацевтическая, Авиационная, Автомобильная, Телекоммуникационная, Ретейл, Энергетика и Коммунальные услуги, др.

Кросс-индустриальный классификатор представляет собой иерархию следующих ключевых сквозных бизнес-процессов (End-to-End):

- R2R – от проводки до отчетности (Record-to-Report);
- H2R – от найма до увольнения (Hire-to-Retire);
- P2P – от запроса до оплаты (Procure-to-Pay);
- O2C – от заказа до оплаты (Order-to-Cash);
- A2R – от получения оборудования до его списания (Acquire-to-Retire);
- F2P – от прогнозирования до планирования продаж и операций (Forecast-to-Plan);
- I2O – от заявки на продукт до оформления заказа (Inquiry-to-Order);
- P2P – от планирования до производства (Plan-to-Produce);
- S2P – от поиска поставщика до оплаты (Source-to-Pay);
- другие

Индустриальный классификатор [15] расширяет кросс-индустриальный классификатор отраслевой спецификой и лучшими практиками конкретной индустрии. Например, для автомобильной отрасли он состоит из операционных процессов (пп. 1-6) и управленческих процессов (пп.7-13).

ОПЕРАЦИОННЫЕ ПРОЦЕССЫ					
1.0 Разработка видения и стратегии Develop Vision and Strategy	2.0 Разработка новых моделей автомобилей Develop New Vehicles	3.0 Маркетинг и продажа новых продуктов/услуг Market and Sell Products and Services	4.0 Создание автомобиля и логистика Build Vehicle and Logistics	5.0 Предоставление услуг Deliver Services	6.0 Предоставление услуг Manage Customer Service
УПРАВЛЕНЧЕСКИЕ ПРОЦЕССЫ					
7.0 Развитие и управление человеческим капиталом			Develop and Manage Human Capital		
8.0 Управление информационными технологиями (ИТ)			Manage Information Technology (IT)		
9.0 Управление финансовыми ресурсами			Manage Financial Resources		
10.0 Приобретение, строительство и управление активами			Acquire, Construct and Manage Assets		
11.0 Управление корп. рисками, комплаенс и устойчивость			Manage Enterprise Risk, Compliance, Remediation & Resiliency		
12.0 Управление внешними связями			Manage External Relationships		
13.0 Развитие и управление бизнес-возможностями			Develop and Manage Business Capabilities		

Рисунок 1 – Классификатор типовых процессов (Automotive PCF® v7.0.5) [6]

Потенциал использования референтных моделей для достижения кардинальных улучшений заключается в проведении нестандартных сравнений и поиске идей, которые обычно не встречаются в рамках рыночной деятельности предприятий. Для обеспечения этого полезного бенчмаркинга референтная модель служит в качестве высокоуровневой, отраслево-нейтральной модели корпоративных процессов, которая позволяет организациям видеть свои бизнес-процессы с точки зрения отраслевой и кросс-отраслевой перспектив.

Цифровизация бизнес-процессов

Важно не только построить целевую процессную модель предприятия, но и достичь высокого уровня её цифровизации и эффективности. В зависимости от уровня развития конкретного предприятия набор ИТ-решений и стоимость их внедрения и использования будут существенно различаться.

Рассмотрим 3 основных этапа развития предприятия:

- 1) **Начальный этап развития, или низкий уровень профессионализма.** На этом этапе компания не имеет существенного ИТ-бюджета, поэтому использует широко используемые ИТ-решения – электронные таблицы, системы ведения бухгалтерского и налогового учета, управления кадрами и расчета заработной платы.
- 2) **Высокий уровень развития и средний уровень профессионализма.** На этом этапе растут затраты на управление предприятием и выполнение бизнес-процессов, поэтому основной задачей является цифровизация бизнес-процессов с целью эффективного управления затратами компании и внедрения системы внутренних контролей. Основными ИТ-решениями для достижения целей компании являются системы управления предприятием класса ERP, DMS и CRM.
- 3) **Передовой уровень развития и высокий уровень профессионализма.** На этом этапе компания уже имеет высокий уровень цифровизации, ставит цели по повышению выручки и получения конкурентных преимуществ. Ключевыми задачами, стоящими на этом этапе, являются:
 - a. **Реинжиниринг текущих бизнес-процессов** – поиск путей оптимизации бизнес-процессов и создания ценности товара для потребителя (lean management, process mining, др.).
 - b. **Автоматизация трудоемких повторяющихся операций** – снижение затрат и времени на проведение транзакционных процессов (OCR, Robotics, др.).
 - c. **Управление данными** – управление данными предприятия как активом и различная бизнес-аналитика, в том числе и нефинансовая (DMBS, Data lake, Data Mesh, Vector DB, др.).
 - d. **Управление принятием решений** – обеспечение бизнеса наиболее продвинутыми способами получения информации, аналитика, прогнозирование для принятия взвешенных решений (Predictive analytics, Machine learning, AI, др.).

Выбор эффективного стека технологий является ключевой задачей цифровизации процессов. Подход к выполнению данной цели определен в Фреймворке цифровой трансформации [1], в котором были сформулированы 5 доменов цифровой трансформации (Клиенты, Платформы, Данные, Эксперименты, Ценность).

Продуктовая организационно-функциональная модель предприятия

Внедрение современных технологий также требует гибкости и адаптивности предприятия, что ведет к развитию организационной структуры предприятия от **функционально-матричной** к **продуктовой** организационно-функциональной модели. Данная модель позволяет:

- 1) быстро реагировать на изменения требований бизнеса (все изменения – в рамках продуктов);
- 2) наращивать численность ИТ-персонала в геораспределенных центрах компетенций и ресурсных пулах;
- 3) повышать скорость разработки за счет использования внутреннего ИТ-персонала, продуктового подхода и внедрения культуры/принципов Agile.

Продуктовая организационно-функциональная модель предполагает организацию подразделений предприятия вокруг продуктов: как внешних, так и внутренних. Такая модель позволяет перестроиться и отразить атаки прорывных инноваторов за счет решения следующих основных задач:

- 1) повышение бизнес-эффективности при работе автономных и кросс-функциональных команд;
- 2) повышение эффективности внедрения современных ИТ-решений;
- 3) единая ответственность за прибыльность продуктов на владельцах этих продуктов;
- 4) грамотное распределение ресурсов без ущерба для текущей деятельности;
- 5) сближение (партнерство) бизнеса и ИТ – повышение уровня технологической зрелости.

В результате предприятие получает конкурентное преимущество во время прорывных инноваций, трансформирующих окружающий мир [16].

На рисунке ниже изображена продуктовая организационно-функциональная модель условного предприятия на основе опыта авторов, состоящая из трех главных частей:

- 1) функциональная структура поддерживающих функций – инфраструктура, бухгалтерия, финансы, др.
- 2) продуктовая структура бизнес-функций – производство и ремонты, склад и логистика, продажи и клиентский сервис, др.
- 3) инновации и технологии – офис управления проектами, лаборатории по инновациям, управление данными, др.

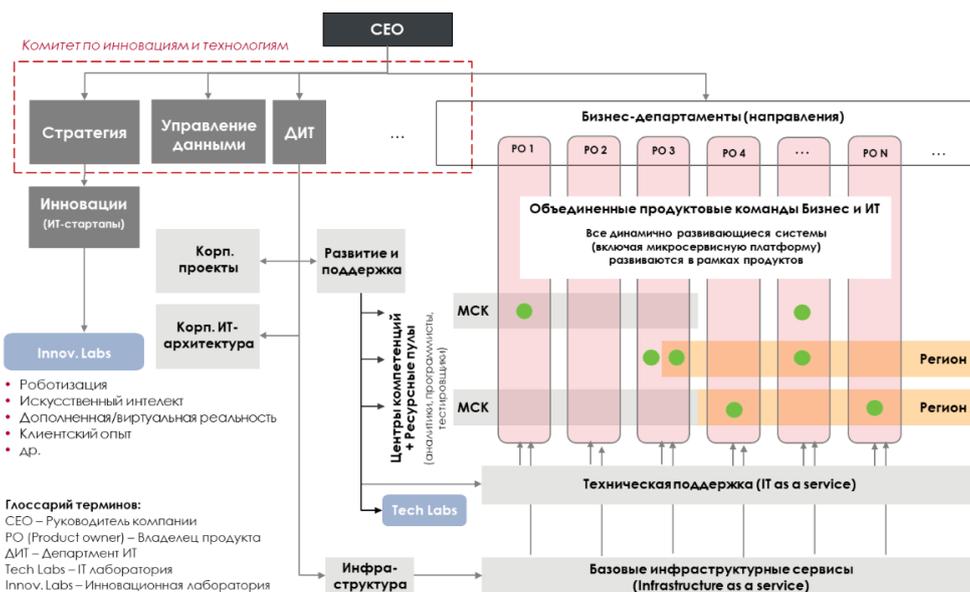


Рисунок 2 – Пример продуктовой организационно-функциональной модели (составлено автором).

Для внедрения продуктовой организационной структуры одной из самых популярных моделей стала тримодальная концепция [17], описывающая управленческую деятельность организации как включающую три сферы:

- 1) операционная (Run) – текущая операционная деятельность предприятия;
- 2) изменения (Change) – деятельность по созданию и внедрению новых решений и продуктов, в большинстве основанных на ИТ;
- 3) «прорывная» (Disrupt) – деятельность по поиску революционных изменений и инноваций, тестированию совершенно новых бизнес-моделей и стартапов внутри большого предприятия.



Рисунок 3 – Тримодальная концепция [8].

Концепция IT4IT

При внедрении продуктовой организационно-функциональной модели ключевую роль занимает управление ИТ, которое является связующим звеном между продуктовыми и функциональными командами.

Термин «Управление ИТ» возник более 20 лет назад и вообрал в себя целый ряд всемирно известных стандартов и лучших практик: ITIL, CISA, TOGAF, PMBoK, RUP и др. Данные стандарты использовались компаниями в качестве ориентира, и специалисты всегда старались собрать их воедино.

В 2014 году промышленным консорциумом по стандартизации и сертификации технологий The Open Group была выпущена первая публичная версия IT4IT – открытого стандарта и референтной архитектуры управления ИТ [18].

ИТ4ИТ (ИТ для ИТ) – это концепция, которая описывает инструменты, процессы и системы, используемые ИТ-организациями для управления своими собственными ИТ-операциями. Эта концепция включает все, что необходимо для планирования, разработки, доставки, эксплуатации и поддержки ИТ-сервисов.

В основу ИТ4ИТ легло понятие цепочки ценности (value chain), описанное Майклом Портером в его книге «Конкурентное преимущество» [19].

Вся деятельность ИТ-организации была разделена на четыре основных потока создания ценности (value stream) и пять вспомогательных видов деятельности. Четыре потока создания ценности объединены и детализированы в виде референтной архитектуры управления ИТ:

- 1) от стратегии к портфолию (Strategy to Portfolio) – планирование (Plan);
- 2) от требования к развертыванию (Requirement to Deploy) – Построение (Build);
- 3) от запроса к исполнению (Request to Fulfill) – выпуск (Deliver);
- 4) от обнаружения к корректировке (Detect to Correct) – запуск (Run).

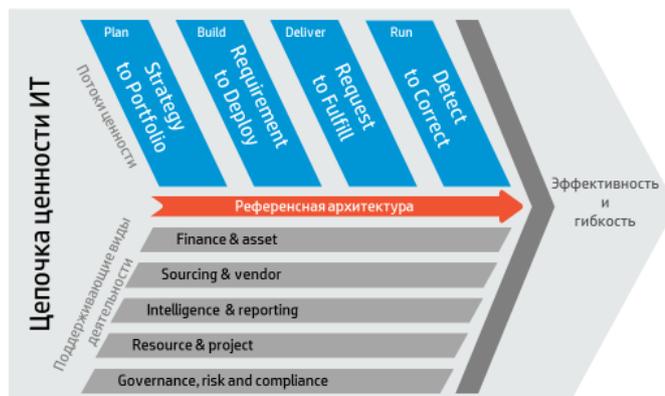


Рисунок 4 – Концепция ИТ4ИТ [20].

Для каждого потока описаны назначение, ценность для бизнеса, KPI, основные объекты данных (information model) и функциональные компоненты (functional model), которые их поддерживают. Все вместе, четыре потока позволяют ИТ контролировать модель услуги (service model) на протяжении жизненного цикла.

Заключение

Резюмируя, цифровая трансформация – не просто задача по использованию современных технологий, а построение эффективной организационно-функциональной модели и целевых процессов на основании лучших практик, которые позволяют эффективно вести бизнес сегодня и быстро реагировать на изменения завтра.

Различные внешние факторы и глобальные тренды оказывают существенное влияние на бизнес-цели и актуальность используемых ИТ-решений. Компании вынуждены ежегодно если и не изменять, то адаптировать стратегию к современным вызовам и инновациям.

Международные и отечественные ИТ-решения содержат определённый набор функций и практик их использования, которые зачастую только частично покрывают потребности предприятия. Поэтому вендоры идут по пути создания гибких механизмов доработки своих решений либо создания решений с использованием открытого кода.

Российские компании, которые занимаются технологическим суверенитетом и хотят сохранить/получить лидерство на рынках, должны обратить внимание на современные подходы по построению эффективной модели цифрового предприятия перед тем, как внедрять те или иные ИТ-решения.

Список использованных источников

1. Фреймворк цифровой трансформации. ИТиММ-2024. Сборник статей XIII Международной научно-практической конференции имени А.И. Китова. Москва, 2024. С. 250-255.
2. Кадры для цифровой трансформации торгово-промышленных предприятий. Шаг в будущее: искусственный интеллект и цифровая экономика. Сборник научных статей VI Международного научного форума. Москва, 2024. С. 36-40.
3. <https://investorday.sber> (дата обращения: 14.09.2024).
4. <https://ir.yandex.ru/> (дата обращения: 14.09.2024).
5. <https://corp.ozon.ru/> (дата обращения: 14.09.2024).
6. Прохоров А., Коник Л. Цифровая трансформация. Анализ, тренды, мировой опыт. – М. : ООО «КомНьюс Групп», 2019. – 368 с.
7. <https://uacrussia.ru/ru/corporation/> (дата обращения: 14.09.2024).
8. <https://www.investopedia.com/articles/personal-finance/121714/worlds-top-10-software-companies.asp> (дата обращения: 14.09.2024).
9. https://www.cnews.ru/reviews/erp_2024/review_table/f6a35eda077385e29ecab9d5020daa86caaa0f59 (дата обращения: 14.09.2024).
10. <https://pro.rbc.ru/demo/658538c49a794741b8bb0469> (дата обращения: 18.11.2024).
11. <https://pro.rbc.ru/demo/6735b3f49a794779723d55ab> (дата обращения: 18.11.2024).
12. Pulse of the Profession 2023. Project Management Institute.
13. О.В.Китова, Брусин С.Н., Дьяконова Л.П. Цифровой бизнес. Инфра-М. 2023. – 418 с.
14. <https://www.apqc.org/> (дата обращения: 14.09.2024).
15. Automotive process classification framework, Version 7.0.5.

16. Зона победы. Управление в эпоху цифровой трансформации / Джеффри Мур ; пер. с англ. Эльвиры Кондуковой. – М. : Манн, Иванов и Фербер, 2020. – 268с.
17. <https://lean.cdto.ranepa.ru/2-1-vidy-upravlencheskoj-deyatelnosti> (дата обращения: 14.09.2024).
18. <https://www.opengroup.org/it4it> (дата обращения: 14.09.2024).
19. Конкурентное преимущество: как достичь высокого результата и обеспечить его устойчивость Майкл Е Портер; Пер. с англ. – М.: Альпина Бизнес Букс, 2005. - 715 с.

Ключевые слова:

цифровая трансформация, референтные модели, бизнес-процессы, организационно-функциональная модель предприятия, продуктовая модель предприятия, концепция IT4IT

Никитчук Сергей Сергеевич,

аспирант, РЭУ им. Г.В. Плеханова (115054, Россия, Москва, Стремянный переулок 36, корпус 9),
ORCID: 0009-0006-3422-6456,
snikitchuk002@yandex.ru

Китова Ольга Викторовна,

доктор экономических наук, доцент, зав. кафедрой информатики РЭУ им. Г.В. Плеханова (115054, Россия, Москва, Стремянный переулок 36, корпус 9),
ORCID: 0000-0002-1820-0954,
kitova.ov@rea.ru

Заявленный вклад соавторов

Китова О. В. – научное руководство, формирование основной концепции статьи и критический анализ; **Никитчук С.С.** – формирование результатов исследования, сбор и обработка информации, выводы.

Sergei S. Nikitchuk, Olga V. Kitova Business Model for Digital Transformation Framework

Keywords:

digital transformation, reference models, business processes, company organizational and functional model, company product model, IT4IT concept.

DOI: 10.34706/DE-2025-01-05

JEL classification: C02 – C4 Эконометрические и статистические методы: специальные темы; C45 Нейронные сети и смежные темы

Abstract

Introduction. The article provides an overview of organizational and process issues of digital transformation of industrial and trade companies. Approach is defined to perform a high-quality and effective digital transformation and build an innovative business model.

Methods. The research method is the selection of modern approaches in the field of transformation of company organizational and process model, which help large companies not only to perform the transformation, but also to make a disrupt using IT technologies as a key driver of change.

Results and Discussion. The main result of the work is an approach to conducting high-quality and effective digital transformation and building an innovative business model. The author's experience and numerous business cases of foreign (Apple, Netflix, Amazon, etc.) and domestic companies (Sber, X5, Ingosstrakh, etc.) show that companies are constantly looking for using of modern technologies, which requires rapid implementation of ideas from initiation to implementation (MVP – minimum viable product) and to quick launch of product to the market. Speed is the main competitive advantage and also a limitation. Each individual product can be both – disrupt and failure. So it is necessary to quickly and cost saving test hypotheses at each stage of product creation (Stage-Gate model).

Companies differentiate technologies for two groups: (1) bring added value (operational) and (2) automated management or regulatory processes. The first ones are often unique developments (R&D), while the second ones are specific to market or industry (ERP, CPM, etc.). They are interconnected and integrated with each other. Therefore, an enterprise should invest in the first ones and save costs for second ones.

Conclusion. The most important conclusion of this article is that Russian companies, that are engaged in technological sovereignty and want to maintain/gain leadership in the markets, should pay attention to modern approaches to building an effective model of a digital enterprise before implementing certain IT solutions.

1.6. Как искусственный интеллект разрушит экономику и уничтожит человечество

Терелянский П. В. д.э.н., к.т.н., профессор,
ФГБОУ ВО Российский экономический университет им. Г.В. Плеханова
г. Москва, Российская Федерация

Статья посвящена анализу потенциальных негативных последствий массового внедрения и бесконтрольного использования технологий искусственного интеллекта. Рассматриваются три основных неблагоприятных сценария. Первый сценарий описывает возможность массовых техногенных катастроф, вызванных непрозрачными алгоритмами. Второй сценарий связан с гносеологической катастрофой, обусловленной информационным шумом, генерируемым ИИ. Третий сценарий предполагает экономическую катастрофу, которая может возникнуть в результате интеллектуализации средств производства. В статье не рассматриваются способы смягчения описанных угроз, автор предлагает непрерывно мониторить возникновение новых технологических, экономических и социальных рисков, порождаемых дальнейшим прогрессом в области внедрения технологий искусственного интеллекта. Своевременное прогнозирование и управление этими рисками позволят снизить вероятность негативных эффектов и направить развитие технологий в позитивное русло.

1. Введение

Помимо очевидных плюсов [Яковлева и др, 2023; Невмывако, 2020], которые приносит развитие и внедрение любых новых технологий, возможны и негативные последствия бесконтрольного и широкого использования искусственного интеллекта. Возможны три основных неблагоприятных сценария и различные их комбинации. Сценарии даны в порядке возрастания их опасности и сложности нивелирования последствий.

Первый сценарий – массовые техногенные катастрофы, порожденные непрозрачными алгоритмами, на которых строятся технологии искусственного интеллекта.

Второй – катастрофа гносеологическая, порожденная информационным шумом, создаваемым искусственным интеллектом.

Третья – катастрофа экономическая, порожденная интеллектуализацией средств производства.

Техногенные катастрофы, порожденные непрозрачными алгоритмами работы AI. Говоря о возможности и причинах массовых техногенных катастроф, особой опасностью которых будет являться их абсолютная непредсказуемость, следует разобраться, что же понимается в массовом сознании под термином «искусственный интеллект».

Существует много различных описаний и сложных академических определений [Яковлева и др, 2023]. Сейчас это в некотором роде «коммерческий», «журналистский» термин для широкой публики, наподобие термина «электронные мозги», который использовали ещё со времен ЭНИАКа.

Сегодня под понятием «искусственный интеллект» в самом широком смысле скрывается несколько основных существующих различающихся между собой технологий: нейронные сети, экспертные системы и автоматическое управление. В самом упрощенном варианте эти технологии решают и разные задачи: нейронные сети – это, прежде всего, классификация объектов, экспертные системы – это построение последовательности логических выводов, автоматическое управление – интерактивное управление технологическими процессами. Понятно, что все эти задачи могут быть сведены воедино в одну систему, которая и будет «выполнять творческие функции, которые традиционно считаются прерогативой человека». Именно такое определение дал советский и российский кибернетик Д.А. Поспелов (с соавторами Аверкиным А.Н. и Гаазе-Рапопортом М.Г.) в 1992-м году в «Толковом словаре по искусственному интеллекту» [Аверкин, А.Н. и др., 1992]. Конкретные «коммерческие» продукты на основе AI содержат в себе еще много различных тоже очень сложных алгоритмов, которые либо готовят данные для AI, либо используют результаты его срабатывания. Например, системы генеративного искусственного интеллекта, создающего картинки по тестовым запросам пользователей, содержат помимо алгоритмов интеллектуального распознавания текста в контексте, ещё и алгоритмы морфинга, алгоритмы моделирования поверхностей, алгоритмы генерации окружения (зданий, деревьев, неба и прочего), которые никогда не относились и посторонней публикой, и инженерами к направлению «искусственный интеллект».

Впервые же термин «Artificial Intelligence» применил Д. Маккарти в 1956 году на конференции в Дартмутском университете. Под искусственным интеллектом в широком смысле подразумевается способность искусственного (созданного человеком объекта) вести себя как естественный (созданный природой за миллионы лет эволюции) субъект. Отметим, что в английском языке есть принципиальные различия между словами «Intelligence» и «Intellect», которые на русский язык принято переводить одним словом – интеллект. Однако «Intelligence» – это прежде всего способность к собиранию и сложной обработке данных. Например, Intelligence Service – служба разведки, шпионажа. То есть Artificial Intelligence в понимании

Д. Маккарти – это прежде всего сложные искусственные системы сложной обработки сложных наборов данных.

Исследование множества определений термина «Artificial Intelligence» и технологий [Невмывако 2020; Хэвен, 2019], которые скрываются за этим описанием, позволяют дать собственное определение: Искусственный интеллект (ИИ, Artificial Intelligence, AI) – алгоритм, способный самостоятельно создавать алгоритмы, в результате срабатывания которых возможна постановка и достижение наиболее оптимальным способом новой, ранее не сформулированной создателем алгоритма цели [Терелянский, 2018a].

Важно отметить, что технологии AI настолько усложнились, что сама готовая рабочая и эксплуатируемая система AI превращается в некий «черный ящик», а сами инженеры-разработчики зачастую не могут предсказать, какие именно результаты будут получены системой AI. То есть результат моделирования срабатывания этого алгоритма другими подобными и даже идентичными алгоритмами не может быть четко определен с приемлемой точностью и вероятностью за время, меньшее или равное времени достижения цели моделируемым алгоритмом.

Особенно это касается нейронных сетей. По сути, всякая сетевая структура с точки зрения программиста представляется матрицей связности, в узлах которой будут либо индикаторы связности, либо веса связности. И обучение сети в итоге идёт как настройка изменений этих весов. Если смотреть глубже, с точки зрения аппаратной реализации, то и матрица связности не матрица, а линейный периодический массив. Математика нейронных сетей устроена так, что непохожие сети с непохожими матрицами могут выдавать в принципе похожие решения. И эти решения будут удовлетворять нас вплоть до наступления некоторого критического события – катастрофы, появления которой мы не можем предугадать. То есть нейронная сеть представляет собой огромную матрицу чисел, понять, что и как зашифровано в ней (на каких примерах настраивались веса) не представляется возможным в принципе.

Например, на одной конференции по AI инженеры системы безопасности рассказали следующее. Разрабатывалась система, которая определяет, можно ли различным автомобилям въезжать на охраняемую территорию. Создатели системы вложили в обучающие примеры «троянского коня» – система однозначно пропускала любую машину, если на капоте была особая надпись-пароль, известная только создателям. Понятно, что это делалось с благими целями – чтобы была возможность проезда машины для ремонта в случае всяких сбоев системы. Но на мой вопрос: «А можете ли вы дать гарантию, что подобную же картинку с собственным паролем не «скормили» нейросети злоумышленники?», инженеры-разработчики не смогли дать однозначного ответа. Это первая и глобальная опасность технологий, построенных на обучаемых нейронных сетях – принципиальная невозможность определить, что же за данные были использованы в обучении.

Вторая фундаментальная опасность нейронных сетей заключается в отсутствии математически четкого, теоретически и практического обоснованного правила, регламентирующего количество обучающих примеров для сети. Например, если у нас будет нейронная сеть, задачей которой будет только дать ответ, человеческое ли это лицо или нет, и мы обучим эту сеть всего на одной фотографии человека, то сеть будет распознавать как человека изображение только на этой фотографии, всё остальное человечество будет отринуто. Если же мы обучим сеть на всех фотографиях человечества, включая, безусловно, людей разумных, но инвалидов или с тяжелыми генетическими заболеваниями и уродствами, то сеть будет распознавать любой более-менее овалный объект как лицо человека. Это примерно, как накладывать полупрозрачные слайды с фотографиями лиц друг на друга. На некотором количестве слайдов можно будет получить усредненную размытую картинку, где четко будут выявлены темные пятна в области глаз и рта, выпуклости по бокам головы – уши, облако волос сверху и так далее. Но если продолжать накладывать слайды, то рано или поздно мы получим размазанное неопределённое пятно, согласно которому за человеческое лицо можно будет даже принять фотографию собаки в полный рост в профиль.

Экспертные системы, которые тоже относятся к сфере искусственного интеллекта, работают по другому принципу. В базу знаний такой системы заносятся наборы правил вида «Если А, то В, иначе С», «Если В, то D, иначе F». Эти правила связаны между собой по переменным А, В, С, D и с помощью логических выражений И, ИЛИ, НЕ. Например, «Если идет дождь, то возьмите зонт» И «Если взяли зонт, то наденьте сапоги». В итоге получается составное правило, которое говорит нам, что «Если идет дождь, то возьмите зонт и наденьте сапоги», хотя в первоначальных правилах связь с дождем имел только зонт. Так экспертные системы получают новые знания и правила. В отличие от нейронных сетей такие системы достаточно прозрачны для понимания срабатывания и контроля логических выводов.

Фундаментальная проблема, из-за которой экспертные системы такого типа не получили широкого распространения – сложность построения и сбора правил [Терелянский, 2009]. Всегда остается большая вероятность того, что необходимые базовые правила не собраны или построены таким образом, что будет невозможно создание необходимых логических цепочек. Если же собрано достаточно большое количество правил, то такие системы начинают сбоить из-за того, что топология связей между правилами становится слишком сложной, а в сложной топологии есть большая вероятность того, что будет построена абсолютно логически связанная система выводов, которая в итоге даст общую бессмысленную рекомендацию. Например, если идёт дождь, то вполне можно построить логическую цепь, которая однозначно будет утверждать, что надо прочистить ливневую канализацию, и ни словом не упомянет про зонт.

Повторюсь, что, говоря об искусственном интеллекте, следует понимать – в текущем прикладном аспекте это прежде всего маркетинговый термин, за которым скрываются сложные управляющие алгоритмы.

Причем с точки зрения проводимых вычислений это могут быть совсем простые операции – что-то типа подсчета средних значений и сравнения их между собой. А вот логика ветвления этих алгоритмов может быть очень сложной. Эти ветвления представляют собой не просто некоторую дорогу с перекрестками, а сложную дорожную сеть, причем сеть в многомерном пространстве. И в этой сложной дорожной сети из точки А в область точки В можно прийти совершенно разными путями. Конкретный путь каждого алгоритма может зависеть от множества мимолетных входных значений и условий, вплоть до скачков напряжения в электрической сети во время вычислений. Поэтому один и тот же алгоритм при одних и тех же входных данных может выдавать разные по оптимальности решения. И просчитать эти решения на таком же точно алгоритме, запущенном параллельно, становится невозможным. То есть, и это особенно важно, алгоритмы AI, из-за собственной сложности и из-за сложности в постановке и определении качества результата, ведут пользователя не точно «из точки А в точку В», а ведут в «область» точки В. Площадь этой области и определяет точность алгоритма, но проблема в том, что пользователь сам не в состоянии определить, где же эта точка В, и вынужден на веру принимать решение, предложенное алгоритмом AI.

Ещё одна сфера, относящаяся к искусственному интеллекту – системы автоматического управления различными технологическими процессами, от выплавки стали до рисования картин и удержания равновесия двуногими роботами. Как правило, такие системы работают, используя следующий принцип: есть выходные значения, которые мы должны достичь, и есть множество входных параметров, которые можно изменять, чтобы достичь нужных выходных значений. Как видим, система закольцовывает выход на вход через какую-то функцию отрицательной обратной связи [Певзнер, 2002]. Задачей AI-инженеров является построение самой оптимальной петли обратной связи.

Проблемы современных систем управления заключаются в том, что управляемые объекты становятся невероятно сложными [Терелянский, 2009]. Сложные объекты порождают огромное количество данных, которые либо невозможно учесть – датчики не успевают срабатывать, либо невозможно обработать – не справляются вычислительные алгоритмы. И сложность объектов будет только возрастать со временем.

Сам фактор времени, не как дискретно-непрерывных отрезков квантования для управления, а Времени как сути, задающей давность, непрерывную протяженность причинно-следственных связей нашей технологической цивилизации, начинает всё больше играть зловещую глубинную, еще не понятую нами роль. Дело в том, что практически все современные технологические решения, которые кажутся нам абсолютно новыми и созданными вот только что, на самом деле базируются на некотором предыдущем технологическом фундаменте. Например, вы приносите с собой абсолютно новый ноутбук и втыкаете его в старую искрящую розетку в своем офисе, расположенном в историческом здании в центре города. Проводка и розетка в этом здании были смонтированы в прошлом, а то и в позапрошлом веке. Подключив к ноутбуку самую современную аппаратуру управления, вы неожиданно для себя интегрируете в эту систему управления и древнюю розетку, и древнюю проводку со всеми наводками и утечками. Через некоторое время розетка выйдет из строя, и в итоге сгорит вся самая современная система управления. Так же обстоит дело и набором обрабатываемых данных – зачастую уже невозможно определить, откуда берутся данные, кем и когда они были подготовлены и какими алгоритмами они обрабатываются, прежде чем попадут в контуры управления.

Так, например, проявилась «Проблема 2000». Суть её заключалась в том, что многие алгоритмы для вычисления дат использовали только две последние цифры в обозначении года, то есть год 2001 с точки зрения программы был раньше, чем 1999-й. Программисты 1960-70 годов честно делали свое дело и не предполагали, что их программы проживут в управляющих контурах (в данном случае – это набор различных аппаратных средств разного года выпуска и разных производителей) больше 10-15 лет. А как выяснилось – управляющие контуры, то есть тот самый искусственный интеллект – могут успешно работать совместно и 50 лет, и дольше. В 2000-м году специалисты сделали всё возможное, чтобы избежать техногенных катастроф [Распоряжение Президента РФ, 1999].

Впереди нас ожидает «Проблема 2038» – данная проблема затронет программы и системы, в которых используется представление времени по стандарту [POSIX](#), или UNIX-время. UNIX-время - это количество [секунд](#), прошедшее с [полуночи 1 января 1970 года](#). А за термином «UNIX-время» скрывается операционная система Android – основа большинства современных интеллектуальных гаджетов.

Ещё один практически необнаруживаемый троянский конь создан в том числе и основной парадигмой объектно-ориентированного программирования (ООП). Главными столпами ООП, помимо классов и абстракции, являются инкапсуляция, наследование и полиморфизм. Инкапсуляция скрывает от программиста суть алгоритма, наследование не позволяет разобратся, какой из алгоритмов в итоге сработает, а полиморфизм вообще разрешает использовать методы объектов с одинаковой сигнатурой совершенно отличным от начального проекта способом. Помимо бестолковой громоздкости генерируемых таким способом исполняемых кодов, мы получаем проблему совершенной невозможности разобраться в дебагируемой программе просто из-за её невероятного объема.

Все перечисленные выше проблемы, имеют одну суть – сложность алгоритмов искусственного интеллекта порождает принципиальную невозможность внешнего (человеческого) контроля за производимыми вычислениями, а следовательно, возникают различные опасные ситуации. Так, например, было с катастрофами Boeing 737 MAX авиакомпании Lion (октябрь 2018 года) и Ethiopian Airlines (март 2019 года). Тогда вину свалили на низкооплачиваемых индийских разработчиков из компаний HCL Technologies и

Cyient. Те, в свою очередь, переложили вину на некачественные спецификации алгоритмов (техническое задание на разработку и тестирование) от Boeing. В итоге всех тяжб бытует мнение, что во всем виноват некий «искусственный интеллект». Но суть проблемы «плохих кодов» и «плохих тестировщиков» заключается на самом деле в том, что плохой громоздкий алгоритм порождает некие неотслеживаемые паразитные ветвления в алгоритмах. Плохой код добавляет ошибки срабатывания, что тоже ведет к увеличению ветвлений, а тестировщики просто не могут отследить все возможные комбинации ошибок (их количество превосходит все разумные пределы). То есть готовый алгоритм искусственного интеллекта уверенно проходит тестирование на некие критические показатели в определенном наборе, но как он будет работать при других показателях, отследить и предугадать невозможно.

И это основные техногенные предпосылки к тому, как усложнение технологий искусственного интеллекта может привести к крупнейшим техногенным катастрофам. А поскольку и устранением последствий таких катастроф рано или поздно займется искусственный интеллект, то и нет возможности предположить, как именно будет выглядеть безопасный для человека мир с точки зрения AI. Ведь с точки зрения логики, теории вероятности, и теории обратных связей выражение «Нет человека, нет проблем» является самым оптимальным решением.

Гносеологическая катастрофа (эпистемологическая), порожденная искусственным интеллектом, может пойти по трем основным направлениям: создание вала неструктурированной и неранжированной информации, утеря классических когнитивных функций рядовым человеком, разрушение этической основы цивилизации. Направления перечислены в порядке возрастания опасности и убывания вероятности.

Возвращаясь к определению термина «Artificial Intelligence» и разнице между «Intelligence» и «Intellect», отметим, что широкая публика воспринимает термин AI все же ближе к русскому термину «интеллект». Слово же «Intellect» и на русском, и на английском языке описывает то, что принято называть сознанием, то есть естественной и неотъемлемой способностью человека разумного осмысливать некие абстрактные понятия – этику, мораль, справедливость, любовь. Именно такие способности поспешили вложить фантасты 50-х годов XX-го века в модный тогда термин «электронный мозг», которыми комплектовались вымышленные роботы. А согласно размышлениям фантастов, все роботы обладали зачатками какой-то собственной морали и собственными целями, которые должны были быть ограничены «тремя законами роботехники» Айзека Азимова. И этот феномен тоже требует собственного определения, которое расширяет функциональность и суть Artificial Intelligence до понятия «Искусственное сознание».

Искусственное сознание (Artificial Consciousness, AC) – алгоритм, способный оценить свою самость, способный самостоятельно создавать собственные морально-этические личностные конструкты (ограничения, правила) на основе анализа его жизненного пространства и действовать в рамках построенных конструктов (ограничений, правил) [Терелянский, 2018b].

Таким образом, ключевым отличием искусственного сознания от узкоутилитарного технологического искусственного интеллекта – является способность ставить перед собой Цель. То есть, не просто точку в пространстве-времени-ресурсах, которую надо достичь, но Цель как нечто метафизическое – личное обогащение, мировое господство, познание Вселенной и Бога.

Развитие техногенной цивилизации приводит к созданию человеком нового искусственного субъекта, который, подобно своему создателю, обретает способность ставить перед собой цели и достигать их с помощью методов, недоступных пониманию человека. Этот субъект наделён сознанием, позволяющим ему осознать собственную индивидуальность, способность осознавать свою самость (осознает собственные границы и выделяет себя из окружающего мира), способностью к кооперации с другими сущностями для достижения собственных целей, способностью взаимодействовать с окружающим миром. Способность к конкуренции за критические ресурсы делает такого субъекта потенциальным соперником человеческого мышления, причём всё говорит о том, что он превзойдёт своих создателей в интеллектуальной мощи. Тогда сразу же возникают онтологические вопросы – а не станет ли создаваемый объект равным создателю-субъекту в области моральной и этической? Имеем ли мы право порождать и уничтожать эти объекты, просто выдернув вилку из розетки?

Разница в понимании «Artificial Intelligence» и «Artificial Consciousness» вскрывает фундаментальную ошибку в постановке задачи тестирования искусственного интеллекта по Тьюрингу. Тест Тьюринга предназначен не для индикации алгоритмов «искусственного интеллекта», понимаемых как некие сложные адаптивные вычислительные приемы, а для индикации именно «искусственного сознания». Ошибка заключается в том, что, согласно теории информации, *нечто* только тогда становится информацией, когда есть приемник информации, а источником информации может быть любой объект или проявление Вселенной. Следовательно, качество оценки AI по Тьюрингу определяется не источником информации (исследуемым алгоритмом), а приемником информации – человеком-исследователем.

Алгоритмы генеративного искусственного интеллекта уже сегодня выдают тексты и изображения, которые даже специалисты вряд ли отличат от созданных человеком-художником. И здесь начинается первая гносеологическая опасность – AI может за секунду создать столько текстов и картинок, осознать и дать оценку которым человечество будет не в состоянии никогда. Следовательно, эти бессмысленные интеллектуальные объекты будут влиты в общие базы данных, библиотеки и картинные галереи на правах созданных людьми осмысленных объектов. Следующее поколение генеративного AI будет использовать уже эти образцы в качестве базы для обучения, а поскольку этих искусственных неосмысленных образцов будет в тысячи раз больше, чем естественных текстов, и в миллионы раз больше, чем

истинных шедевров, то в скором времени человечество захлебнется в вале построенного по классическим канонам литературы, но бессмысленного внутренне унылого Pulp Fiction. То есть, на самом деле, генеративный AI работает как известный в математике алгоритм скользящего среднего – постепенно сглаживает всплески гениальности до простой прямой линии бездарности.

Вал усредненной, неструктурированной и неранжированной человеческим гением информации, породит следующую проблему – рядовой человек не будет иметь ярких интеллектуальных и этически выверенных маяков, на которые он бы мог равняться в повседневной жизни и личном творчестве. Мучительный творческий акт гения будет просто похоронен кучей мусорных произведений искусства в том числе за счет экономической составляющей – на что будет жить художник, годами оттачивающий своё искусство и годами же вымучивающий из глубины души истинный шедевр? Не будет шедевров, не будет маяков, не будет развития искусства, изобретательства, технологий. В конце концов, все морфологические ящики [Андрейчиков, 2004] Фрица Цвикки¹ всех областей человеческой деятельности будут заполнены всевозможными комбинациями, оценить разумность которых просто никто не сможет и не сможет выбрать действительно нужную комбинацию. А следом, никто уже не сможет создать новую морфологическую матрицу, которую бы бодро заполнил AI множеством комбинаций действительно нужных человечеству технических или культурных сущностей.

Огромное количество сгенерированных AI объектов будет постепенно заполнять все возможные информационные хранилища, погребая под слоем искусственных объектов результаты осмысленного труда человека. Любая чисто техническая катастрофа или сбой систем хранения информации (которая случится неизбежно и неминуемо) приведёт к тому, что уникальная «естественная» информация невосвратно погибнет, просто по тому, что с точки зрения теории вероятности легче будет восстановить или воссоздать заново искусственные объекты за счет их многочисленности. Эта ситуация знакома многим, кто терял или иным образом утрачивал собственные носители информации – старые файлы и уникальные семейные фотографии утрачивались, а на резервных носителях удавалось найти далеко не всю важную информацию, а вот различный информационный мусор легко был доступен.

На этом фоне совсем мелкой выглядит проблема утраты когнитивных функций по принятию любых решений – от тактических, до стратегических – AI сделает это просто быстрее, но отнюдь не значит, что более оптимально и правильно. Уже сейчас большинство людей с трудом производят в уме простейшие арифметические операции: все пользуются калькуляторами. А уж об извлечении корней или вычислении косинусов на бумаге просто не идет речи – никто не вспомнит алгоритм этого действия. И если алгоритм работы калькулятора будет неверным, то полученные неверные результаты, во-первых, никогда не будут проверены, а во-вторых, станут труднообнаруживаемой основой для ряда других ошибок. Вспомните, когда последний раз вы проверяли расчеты одного калькулятора на другом калькуляторе? Думаю, что до сегодняшнего момента это просто не приходило вам в голову. А ведь такая проблема (ошибка Pentium FDIV) была выявлена в 1994 году случайно, практически через год после начала активной эксплуатации процессоров Pentium фирмы Intel: при делении чисел с плавающей запятой команда FDIV процессора для некоторых комбинаций чисел выдавала ошибочный результат [Джексон, 2013].

Постулат безошибочной работы вычислительной машины, а затем и AI приводит к тому, что человек принимает результат работы AI за истину и использует непроверяемые результаты как основную мотивацию собственных действий. Студенты сейчас формируют реферативные работы не на основе проработки большого объема информации, не на основе собственной оценки её нужности путем трудоемкого сопоставления собственноручно полученных фрагментов текстов с некоей целью исследования, получая при этом навык разумной фильтрации фактов и явлений, а на основе создания краткого запроса к AI. Причем запрос этот они не формируют сами, а получают в качестве задания от преподавателей. Что дает такой навык студенту? Навык осознания многообразия мира? Выстраивания в мозгу собственных уникальных и, возможно, гениальных озарений? Накопление фактов и знаний? Нет – только навык составления запроса к текущей реализации AI. Следующим этапом станет уже создание самих заданий не усилиями разума преподавателя, а путем обращения преподавателей к AI. В итоге, мы снова получим скользящее среднее – вместо биения живого сердца прямую линию асистолии.

Так, постепенно подменяя собой когнитивную функцию принятия решений из-за отсутствия, во-первых, маяков, во-вторых, из-за утраты человеком постоянно тренируемой необходимости анализировать и выбирать, AI полностью переформатирует вкусы человека, мораль и этику в дальнейшем. Ведь именно на основе осознанного трудного и мучительного выбора среди множества соблазнов и ограничений, на основе ярких ценностей и строится мораль отдельного человека и этика общества в целом. И здесь не идет речь о заговоре некоего мирового правительства, которое по своему злоумышлению будет «подкручивать» настройки AI и подменять цели человечества собственными меркантильными. Хотя и эта возможность не только не исключена, но и всюду применяется в алгоритмах формирования лент новостей и интересов социальных сетей и видеохостингов. Нет, это всё будет происходить с математической точностью и неизбежностью – такова суть работы алгоритмов AI.

¹ Формальный «механический» метод, предложен астрофизиком Ф. Цвикки для классификации галактик по внешнему виду, позволяющий получить все возможные комбинации объектов нескольких множеств. Яркий бытовой пример – расписание уборки подъезда: по горизонтали даты уборки, по вертикали – ответственные, по диагонали – осмысленная комбинация «дата-ответственный», все остальные ячейки матрицы не имеют смысла в данном случае.

Разрушение этической основы цивилизации также будет идти и через машинное тьюторство [Стригун, 2014; Дрожжих, 2022] в образовательных учреждениях. Главный риск внедрения образовательных роботов связан с возможностью полной индивидуализации и инкапсуляции учебного процесса. Если ребёнок будет взаимодействовать только с роботом, а не с коллективом сверстников, это может привести к утрате важных социальных навыков. Кроме того, остаётся открытым вопрос о том, каким контентом наполнена такая система и насколько она безопасна для ребёнка, поскольку контроль над этим процессом со стороны учителей и родителей значительно затрудняется. Основная цель человеческого обучения состоит не столько в получении индивидуальных знаний, сколько в развитии способности к коммуникациям и познанию в рамках коллектива. Потенциальное отсутствие взаимодействия с реальными людьми в образовательном процессе может негативно сказаться на формировании этих ключевых компетенций.

Ещё одна проблема заключается в усилении недоверия ребёнка к живым учителям, родителям и взрослым в целом. Согласно педагогическим теориям, в определённом возрасте происходит процесс отделения ребёнка от родителей и других взрослых, известный как «сепарация». В этот период дети становятся особенно восприимчивыми к влиянию сверстников, уличной среды и даже криминальных группировок [Петренко и др., 2014; Петренко и др., 2016]. До недавнего времени AI воспринимался скорее как нечто внешнее по отношению к традиционной школьной педагогике, как альтернатива основополагающим образовательным и нравственным ценностям. Однако с внедрением и признанием школами AI-тьюторов, ребёнок может начать формировать свои моральные и интеллектуальные основы на основе общения с чат-ботами. Под влиянием принципа сепарации доверие к таким системам будет выше, чем к школе и родителям.

Экономическая катастрофа на данном этапе представляется абсолютно непреодолимой и крайне вероятной.

AI неизбежно приведёт к существенному изменению производственной основы базиса (по марксизму). А согласно тому же марксизму, изменение базиса, безусловно, приведет и к изменению всей надстройки. Выдержат ли государства такие изменения, будет ли способна современная капиталистическая система наступать на собственное горло постоянного извлечения прибыли и бросить капиталы на грядущие социальные преобразования?

В 2011 году на промышленной выставке в Ганновере представители правительства Германии заявили о необходимости более широкого применения информационных технологий в материальном (прежде всего) производстве. Внимательный анализ концепции Industry 4.0, предложенной немецким правительством, говорит о том, что под четвертой промышленной революцией имеется в виду в основном широкое внедрение кибер-физических систем (cyber-physical system), а следовательно, и AI во многих технологических процессах и цепочках.

Концепция изменения базиса, основанная на классическом понимании марксистской политической экономии, приводит к важному выводу: из состава производительных сил поступательно и необратимо будут исчезать люди. Интеллектуализация средств производства повлечет за собой радикальные социальные изменения в производственных отношениях. Ранее капиталист владел только собственностью на средства производства, а наемный рабочий владел компетенциями, умениями, способностями, которые были его неотъемлемой неотчуждаемой собственностью и позволяли рабочему добывать средства к существованию. Теперь в условиях интеллектуализации средств производства, у наемного работника не остается неотъемлемых свойств. Его компетенции и умения перестают быть уникальным активом, так как они заменяются автоматизированными системами, управляемыми искусственным интеллектом. Традиционные формы трудовых отношений попросту исчезнут, что ставит под угрозу экономическое положение значительной части общества [Паньшин, 2020] и всю, выстроенную тысячелетиями, экономическую структуру отношений. Более того, исчезает возможность повышения своих личных компетенций – алгоритмы AI способны к обучению на порядки выше и дешевле, способны обобщать опыт групп людей, способны объединять разнородные компетенции, которые не могут объединять в себе люди физически. Например, невозможно быть боксером-тяжеловесом и быть мастером бисероплетения, поскольку человек физически может развивать либо мелкую моторику, либо мощные ударные воздействия разбитых кулаков.

Если создается интеллектуальная система, во многих функциях превосходящая человека, то с точки зрения экономической оптимальности (в рамках отношений работник-работодатель) работодателю выгодней использовать искусственный объект: ему не надо платить заработную плату, обеспечивать социальные гарантии, участвовать в социальных взаимодействиях. И, говоря о выгоде, мы не можем говорить только о наживе, в условиях капиталистической конкуренции внедрение AI просто физически необходимо для выживания фирмы.

Но что будет, если AI окончательно заменит специальности, не требующие высокой квалификации и напряженной интеллектуальной деятельности, и которые уже сейчас начинают заменяться роботами-пылесосами, автоматизированными системами складской логистики и беспилотными такси? Смогут ли массово вытесняемые с этих производственных позиций малообразованные низкокавалифицированные люди встроиться в рынок труда [Моттаева и др., 2020], а следовательно, в социально-экономическую систему государства как надстройки?

Выброшенные из производственных отношений люди будут вынуждены идти по двум путям, чтобы просто не умереть с голода. Первый путь – путь насилия, от возрождения луддистских идей [Ваганов, 2019] до организованного грабежа материальных ценностей в более развитых районах. Второй – полная

социальная и интеллектуальная деградация больших слоев населения [Талалаева, 2020; Евлоева, 2019] в «технологических» гетто [Берберов, 2017].

Неолуддизм, порожденный внедрением AI, будет существенно отличаться от событий Первой промышленной революции. В Англии потерявших работу ткачей просто перевешали или выслали в Австралию, а оставшаяся часть смогла хоть как-то интегрироваться в рынок труда только за счет того, что все остальные профессии были технологически простыми и не требовали высоких компетенций для освоения новых инструментов – любой потомственный высококвалифицированный ткач мог выполнять малоинтеллектуальную работу грузчика на автоматизированном текстильном производстве за еду. Сейчас – в эпоху неолуддизма – высококлассный нейрохирург, заменённый автономным роботом, не сможет стать программистом и настройщиком этого робота – и на программиста, и на нейрохирурга надо учиться десяток лет, и эти профессии никак не пересекаются.

Да и смогут ли профессии развиваться в принципе? Здесь будет следующая последовательность строго экономических событий: чтобы стать высококлассным нейрохирургом, способным создавать новые технологии операций, необходимо долго и за дорого учиться и многие годы иметь медицинскую практику, то есть работать нейрохирургом за деньги. Однако, «снятые» с нейрохирургов компетенции будут переданы роботу с AI [Аннаоразова и др., 2024], конкретный нейрохирург потеряет практику сам, потеряет физическую возможность интеллектуально развиваться, аккумулировать и создавать новые знания, потеряет возможность дать образование своим детям, так как потерял практику (оплачиваемую работу) – петля обратной положительной связи замкнется и начнёт свою катастрофическую работу. На этом развитие компетенций профессии прекратится (смотри рассуждения по поводу проблемы утраты когнитивных функций выше).

Современный путь развития нашей технологической цивилизации подразумевает, что алгоритмы AI на основе BigData должны окончательно децентрализовать процесс принятия стратегических, тактических и оперативных управленческих решений, перенаправив эту задачу киберфизическим системам, вплоть до полного исключения человека из производственного процесса. Предполагается, что высвобожденные из производственного процесса люди перекавалифицируются в наладчиков и контролеров технологических линий, в инженеров и создателей новых технологий.

Но этого не будет из чисто экономических причин. При высвобождении тысячи низкоквалифицированных рабочих с конвейера, вряд ли будет создана ровно тысяча рабочих мест контролеров и наладчиков, поскольку тысяча уволенных низкообразованных разнорабочих одновременно не смогут диверсифицировать свои компетенции до уровня высоко- и узкоспециализированных контролеров и наладчиков. Что же касается преобразования массы рабочих, пусть даже высококвалифицированных, в инженеров-изобретателей и инноваторов, обеспечивающих технологические прорывы, то эта трансформация представляется крайне маловероятной, поскольку прорывное инновационное творчество – удел единиц при условии, если есть постоянная экономическая поддержка работы изобретателя.

С массой безработных людей, как потерявших работу, так и никогда её не имевших из-за внедрения AI, надо будет что-то делать. В эпоху луддизма потеряли работу тысячи, и их удалось жесточайшим репрессиями подавить, а в эпоху неолуддизма безработными станут миллионы, и эти миллионы будет просто невозможно физически уничтожить.

Второй путь – создание «технологических» гетто, где будут находиться миллионы невостребованных голодных людей, не имеющих никаких компетенций – уже опробован на практике и показал самые ужасающие результаты. Так называемые «welfare ghettos» («социальные гетто») показывают, что люди поколениями сидят на пособиях и не желают работать, так как климатические и экономические условия, например США, позволяют решать витальные потребности и бесконтрольно повышать популяцию безработных. Казалось бы – витальные потребности решены, доступ к мировым интеллектуальным ресурсам, к собственному образованию и повышению квалификации возможен за счет ресурсов Интернета, но из социальных гетто никто не может вырваться в массовом порядке. Но США могут себе позволить содержать такие гетто, а в странах Африки – такой возможности нет. Так же там нет и возможности повышать компетенции, а следовательно, нет возможности продавать свой труд, а значит, нет возможности участвовать в экономической системе, чтобы повышать свои компетенции. Целый континент оказался на задворках технологической цивилизации, и нет никакой возможности вырваться из этой петли, а усложнение технологий, внедрение AI ещё более затягивают эту удавку.

Так выглядят три основных неблагоприятных сценария. В этой статье не предлагаются пути нивелирования перечисленных угроз, дается лишь попытка осмысления негативных путей бесконтрольного внедрения технологий искусственного интеллекта. Чтобы избежать реализации этих апокалиптических сценариев, необходимо чётко понимать возможности и риски, которые способны создать внедрение искусственного интеллекта. Прежде всего, следует сосредоточиться на разработке алгоритмов, обеспечивающих прозрачность и контролируемость принимаемых решений. Эти меры помогут предотвратить непредвиденные последствия, связанные с автономией AI. Важно постоянно отслеживать появление новых технологических, экономических и социальных рисков, возникающих вследствие дальнейшего прогресса в этой сфере. Прогнозирование и управление такими рисками позволят минимизировать негативные эффекты и направить развитие технологий в конструктивное русло. Наконец, требуется уделять внимание вопросам этического регулирования и правового контроля применения искусственного интеллекта. Только комплексный

подход, включающий технические, правовые и этические аспекты, позволит обеспечить безопасное и устойчивое будущее в условиях широкого распространения AI.

Литература

1. Распоряжение Президента Российской Федерации от 17.06.1999 г. № 194-рп «О неотложных мерах по решению «Проблемы 2000» в Российской Федерации»
2. Аверкин, А.Н. и др. (1992) Толковый словарь по искусственному интеллекту / Авторы-составители А.Н. Аверкин, М.Г. Гаазе-Рапопорт, Д.А. Поспелов. М.: Радио и связь, 1992. –256с.
3. Андрейчиков, А. В. (2004) Математические модели и программные средства принятия и синтеза решений в экономике и бизнесе : монография / А. В. Андрейчиков М-во образования Рос. Федерации, Волгогр. гос. техн. ун-т. – Волгоград : Политехник, 2004.
4. Аннаорова, А. и др. (2024) Современные Достижения в Нейрохирургической Робототехнике: Будущее Точной Хирургии / А. Аннаорова, Д. Какамырадова, М. Гайтназарова // Научные открытия 2024 : Сборник материалов LV-ой международной очно-заочной научно-практической конференции. В 3-х томах, Москва, 22 ноября 2024 года. – Москва: Научно-издательский центр "Империя", 2024. – С. 37-41.
5. Берберов, А. Б. (2017) Роботизация и технологическая безработица XXI века: возрождение луддизма? / А. Б. Берберов // Московский экономический журнал. – 2017. – № 3. – С. 20.
6. Ваганов, А. Г. (2019) Психология луддизма в XXI веке / А. Г. Ваганов // Энергия: экономика, техника, экология. – 2019. – № 10. – С. 7-11.
7. Джексон, Т. (2013) Inside Intel. История корпорации, совершившей технологическую революцию XX века – М.: Альпина Паблишер, 2013. — 328 с.
8. Дрожжих, Н. В. (2022) Цифровой тьютор в Тюменском госуниверситете: опыт внедрения и использования / Н. В. Дрожжих, И. Е. Белякова // Вестник Московского университета. Серия 19: Лингвистика и межкультурная коммуникация. – 2022. – № 2. – С. 141-151.
9. Евлоева, Ф. Р. (2019) Проблема отчуждения человека в российском обществе и глобальном мире / Ф. Р. Евлоева // Вестник Бурятского государственного университета. – 2019. – № 2. – С. 74-82.
10. Моттаева, А.Б. и др. (2020) Влияние искусственного интеллекта на рынок труда / А.Б. Моттаева, В.Л. Кашинцева, О.Ю. Покровский // Вестник Московского государственного областного университета. Серия: Экономика. – 2020. – № 4. – С. 82–88.
11. Невмывако, В. П. (2020) Всеобщая цифровизация и искусственный интеллект в системе управления стратегическим развитием России в условиях перехода в Индустрию 4.0 / В. П. Невмывако // Экономика и социум: современные модели развития. – 2020. – Т. 10, № 4. – С. 331-344.
12. Паньшин, И. В. (2020) Эффект цифровизации при отчуждении труда: новый взгляд на формирование информационного капитала / И. В. Паньшин, И. Ф. Жуковская // Экономика труда. – 2020. – Т. 7, № 10. – С. 867-886.
13. Певзнер, Л. Д. (2002) Теория систем управления : Учебное пособие / Л. Д. Певзнер. – Москва : Московский государственный горный университет, 2002. – 469 с.
14. Петренко, Т. В. и др. (2014) Возрастные этапы процесса сепарации в русле психоаналитических теорий / Т. В. Петренко, Л. В. Сысоева // Актуальные вопросы развития науки : сборник статей Международной научно-практической конференции: в 6 частях, Уфа, 14 февраля 2014 года – Уфа: Башкирский государственный университет, 2014. – С. 110-112.
15. Петренко, Т. В. и др. (2016) Возрастные этапы процесса сепарации от родителей / Т. В. Петренко, Л. В. Сысоева // Акмеология. - 2016. - № 4 (60). - С. 180-185
16. Стригун, А. И. (2014) Компьютерные интеллектуальные тьюторы / А. И. Стригун // Образовательные технологии (г. Москва). – 2014. – № 4. – С. 99-108.
17. Талалаева, Е. Ю. (2020) Этноконфессиональные иммигрантские гетто как проблема национальной безопасности в современном общественно-политическом дискурсе Дании / Е. Ю. Талалаева, Т. С. Пронина // Балтийский регион. – 2020. – Т. 12, № 3. – С. 55-71.
18. Терелянский П.В. (2018а) Цифровая экономика, искусственный интеллект, индустрия 4.0 // Тенденции развития Интернет и цифровой экономики / Труды I Всероссийской с международным участием научно-практической конференции, Симферополь-Алушта, 29-31 мая 2018. - с.91-96.
19. Терелянский, П. В. (2009) Непараметрическая экспертиза объектов сложной структуры : монография / П. В. Терелянский ; П. В. Терелянский ; Изд.-торговая корпорация «Дашков и К°». – Москва : Дашков и К°, 2009.
20. Терелянский, П. В. (2009) Системы поддержки принятия решений. Опыт проектирования : монография / П. В. Терелянский Федеральное агентство по образованию, Волгоградский гос. технический ун-т. – Волгоград : ВолгГТУ, 2009.
21. Терелянский, П. В. (2018b) Искусственный интеллект в индустрии 4.0 / П. В. Терелянский // Цифровая экономика. – 2018. – № 3(3). – С. 42-49.
22. Хэвен, Д. (2019) Искусственный интеллект. Что надо знать о наступающей эпохе разумных машин/ пер. с англ. О. Д. Сайфудиновой. — М.: АСТ, 2019. 352 с.

23. Яковлева, Е. А. и др. (2023) Роль технологий искусственного интеллекта в цифровой трансформации экономики / Е. А. Яковлева, А. Н. Виноградов, Л. В. Александрова, А. П. Филимонов // Вопросы инновационной экономики. – 2023. – Т. 13, № 2. – С. 707-726.

References in Cyrillics

1. Decree of the President of the Russian Federation dated 06/17/1999 No. 194-rp "On urgent measures to solve the "Problem 2000" in the Russian Federation"
2. Andreychikov, A.V. (2004) Mathematical models and software tools for decision-making and synthesis in economics and business: a monograph / A.V. Andreychikov Ministry of Education of the Russian Academy of Sciences. Russian Federation, Volgogr. State Technical University un-T. Volgograd: Polytechnic, 2004.
3. Annaorazova, A. et al. (2024) Modern Achievements in Neurosurgical Robotics: The Future of Precision Surgery / A. Annaorazova, D. Kakamyradova, M. Gaitnazarova // Scientific Discoveries 2024: Proceedings of the 14th International Correspondence Scientific and practical Conference. In 3 volumes, Moscow, November 22, 2024. Moscow: Empire Scientific Publishing Center, 2024, pp. 37-41.
4. Averkin, A.N. et al. (1992) Explanatory dictionary of artificial intelligence / Compiled by A.N. Averkin, M.G. Haase-Rapoport, D.A. Pospelov. Moscow: Radio and Communications, 1992. -256с.
5. Berberov, A. B. (2017) Robotization and technological unemployment of the 21st century: the revival of Luddism? / A. B. Berberov // Moscow Economic Journal. – 2017. – No. 3. – p. 20.
6. Evloeva, F. R. (2019) The problem of human alienation in Russian society and the global world / F. R. Evloeva // Bulletin of the Buryat State University. 2019. No. 2. pp. 74-82.
7. Haven, D. (2019) Artificial intelligence. What you need to know about the coming era of intelligent machines/ edited by D. Haven: translated from English by O. D. Saifudinova. Moscow: AST, 2019. 352 p.
8. Jackson, T. (2013) Inside Intel. The unauthorized history of the world's most successful chip company. Moscow: Alpina Publisher, 2013. 328 p.
9. Mottaeva, A.B. et al. (2020) The influence of artificial intelligence on the labor market / A.B. Mottaeva, V.L. Kashintseva, O.Y. Pokrovsky // Bulletin of the Moscow State Regional University. Series: Economics. 2020. No. 4. pp. 82-88.
10. Nevmyvako, V. P. (2020) Universal digitalization and artificial intelligence in the management system of strategic development of Russia in the context of transition to Industry 4.0 / V. P. Nevmyvako // Economics and society: modern models of development. – 2020. – Vol. 10, No. 4. – pp. 331-344.
11. Panshin, I. V. et al. (2020) The effect of digitalization in the alienation of labor: a new look at the formation of information capital / I. V. Panshin, I. F. Zhukovskaya // Labor economics. 2020. Vol. 7, No. 10. pp. 867-886.
12. Petrenko, T. V. et al. (2014) Age stages of the separation process in line with psychoanalytic theories / T. V. Petrenko, L. V. Sysoeva // Actual issues of science development : collection of articles of the International Scientific and Practical Conference: in 6 parts, Ufa, February 14, 2014 – Ufa: Bashkir State University, 2014. – pp. 110-112.
13. Petrenko, T. V. et al. (2016) Age stages of the separation process from parents / T. V. Petrenko, L. V. Sysoeva // Acmeology. - 2016. - № 4 (60). - Pp. 180-185
14. Pevsner, L. D. (2002) Theory of control systems: A textbook / L. D. Pevsner. Moscow : Moscow State Mining University, 2002. 469 p.
15. Shivers, N. V. (2022) Digital tutor at Tyumen State University: experience of implementation and use / N. V. Shivers, I. E. Belyakova // Bulletin of the Moscow University. Episode 19: Linguistics and Intercultural Communication. – 2022. – No. 2. – pp. 141-151.
16. Strigun, A. I. (2014) Computer intelligent tutors / A. I. Strigun // Educational technologies (Moscow). - 2014. – No. 4. – pp. 99-108.
17. Talalaeva, E. Y. (2020) Ethno-confessional immigrant ghettos as a problem of national security in modern socio-political discourse in Denmark / E. Y. Talalaeva, T. S. Pronina // Baltic region. – 2020. – Vol. 12, No. 3. pp. 55-71.
18. Tereliansky P.V. (2018a) Digital economy, artificial intelligence, industry 4.0 // Trends in the development of the Internet and the digital economy / Proceedings of the I All-Russian Scientific and Practical Conference with international participation, Simferopol-Alushta, May 29-31, 2018.91-96.
19. Tereliansky, P. V. (2009) Decision support systems. Design experience : a monograph / P. V. Tereliansky Federal Agency for Education, Volgograd State Technical University. Volgograd: VolgSTU, 2009.
20. Tereliansky, P. V. (2009) Nonparametric examination of objects of complex structure : a monograph / P. V. Tereliansky Publishing house-trading Corporation Dashkov and Co.– Moscow : Dashkov and Co., 2009.
21. Tereliansky, P. V. Artificial intelligence in industry 4.0 / P. V. Tereliansky // Digital economy. – 2018. – № 3(3). – Pp. 42-49.
22. Vaganov, A. G. (2019) Psychology of Luddism in the 21st century / A. G. Vaganov // Energy: economics, technology, ecology. – 2019. – No. 10. – pp. 7-11.

23. Yakovleva, E. A. et al. (2023) The role of artificial intelligence technologies in the digital transformation of the economy / E. A. Yakovleva, A. N. Vinogradov, L. V. Alexandrova, A. P. Filimonov // Issues of innovative economics. – 2023. – Vol. 13, No. 2. – pp. 707-726.

Терелянский Павел Васильевич.

д.э.н., к.т.н., профессор,

ФГБОУ ВО Российский экономический университет им. Г.В. Плеханова

г. Москва, Российская Федерация

e-mail: tereliansky@mail.ru

SPIN-код: 7616-8950, ORCID 0000-0003-0642-2901

Ключевые слова

искусственный интеллект, техногенные катастрофы, информационный шум, неолуддизм, экономика данных

Tereliansky P.V., How artificial intelligence will destroy the economy and destroy humanity

Keywords

artificial intelligence, man-made disasters, information noise, neo-Luddism, data economics.

DOI: 10.34706/DE-2025-01-06

JEL classification: F02: F62

Abstract

The article is devoted to the analysis of the potential negative consequences of the mass introduction and uncontrolled use of artificial intelligence technologies. Three main unfavorable scenarios are considered. The first scenario describes the possibility of massive man-made disasters caused by opaque algorithms. The second scenario is associated with an epistemological catastrophe caused by information noise generated by AI. The third scenario assumes an economic catastrophe that may result from the intellectualization of the means of production. The article does not consider ways to mitigate the described threats, the author suggests continuously monitoring the emergence of new technological, economic and social risks generated by further progress in the field of the introduction of artificial intelligence technologies. Timely forecasting and management of these risks can reduce the likelihood of negative effects and direct technology development in a positive direction.

УДК: 657. 658. 65.011.56 .65.02. 65.03

1.7. Исполнение уточненного эконометрического прогноза на 2024 г. ВВП России в условиях специальной военной операции

Афанасьев А.А., д.э.н.
 Финансовый университет при Правительстве РФ, ЦЭМИ РАН, Москва
 Пономарева О.С.
 ЦЭМИ РАН, Москва *В статье рассматриваются.*

Авторами представлены уточненный эконометрический прогноз на 2024 г. темпа роста ВВП России в условиях специальной военной операции, выполненный на основе народнохозяйственной производственной функции, и точность его исполнения. После публикации Росстатом и Мировым Банком фактических данных за 2024 г. оказалось, что значения таких факторов как уровень загрузки производственных мощностей и мировой цены на нефть марки «Юралс» оказались несколько меньше ожидаемых авторами. В результате значения уточненных прогнозных темпов роста ВВП немного снизились, составив 4,1–4,2% против изначально прогнозируемых 5,4–5,5%, и таким образом, почти совпали с фактическим темпом роста в 4,1%, согласно первой оценке Росстата. Основными факторами роста в 2024 г. послужили основные фонды, уровень их загрузки в промышленности, мировая цена на нефть, в то время как уровень занятости оказал на него относительно слабое влияние. Исследованная производственная функция может быть успешно использована для дальнейшего прогнозирования динамики экономического роста в России в обстановке нарастающего санкционного давления со стороны коллективного Запада.

В работе [Афанасьев, Пономарева, 2024] авторами был представлен эконометрический прогноз на 2024 г. ВВП России и темпа его роста в условиях специальной военной операции России на Украине и связанного с ней нарастающего санкционного давления на российскую экономику со стороны правительств стран Запада, не согласных с ее проведением. Прогноз осуществлялся по макроэкономической производственной функции

$$Y_t = e^{\alpha} (n_t Z_t)^{\gamma} V_t^{1-\gamma} e^{\delta O_t}, \quad (1)$$

где Y_t – валовый внутренний продукт России в постоянных ценах 1990 г. в году t , Z_t – среднегодовая стоимость основных фондов экономики России в постоянных ценах 1990 г. в году t , n_t – среднегодовой уровень загрузки производственных мощностей в российской промышленности в году t , V_t – среднегодовая численность занятых в российской экономике в году t , O_t – мировая цена на нефть в долларах США 2010 г. в году t (цена нефти марки «Брент» до 2021 г. включительно, с 2022 г. – цена нефти марки «Юралс»).

Прогноз на 2024 г. был сделан по функции (1) на основе данных таблицы 1 при следующих предположениях о значениях факторов производства в 2024 г. [Афанасьев, Пономарева, 2024].

1. Среднегодовая стоимость основных фондов в российском народном хозяйстве вырастет на 3%, что является средним значением за два предыдущих года.
2. Среднегодовой уровень загрузки производственных мощностей в российской промышленности увеличится и составит 78%.
3. Среднегодовая численность занятых в народном хозяйстве возрастет на 0,7%.
4. Цена на сырую нефть марки «Юралс» составит 60 долл./барр. в долларах США 2010 г., что соответствует номинальной цене 65–70 долл./барр.

При вышеуказанных условиях прогнозные на 2024 г. темпы роста ВВП России по достаточно длинным обучающим выборкам с небольшими средними ex-post прогнозными ошибками (4,1% и 4,2%) составили 5,4–5,5% (Афанасьев, Пономарева, 2024). Такие прогнозные темпы роста ВВП России можно рассматривать как самые оптимистичные из всех известных авторам научно обоснованных прогнозов. Однако по факту в 2024 г. значения факторов производства оказались меньше, чем предполагали авторы.

По итогам 2024 г. фактические значения аргументов макроэкономической производственной функции (1) получились следующими.

1. Среднегодовой уровень загрузки производственных мощностей в российской промышленности составил 77% против ожидаемых авторами 78%. Среднегодовой уровень загрузки мощностей вычислялся путем умножения его прошлогоднего значения (76%) на отношение (1,0141) величины сводного среднегодового уровня загрузки мощностей видов экономической деятельности «Добыча полезных ископаемых, обрабатывающие производства, обеспечение электрической энергией, газом и паром; кондиционирование воздуха» по Росстату (60,92%) к величине этого

показателя за прошлый год (60,07%). В свою очередь, сводный среднегодовой уровень загрузки мощностей видов экономической деятельности «Добыча полезных ископаемых, обрабатывающие производства, обеспечение электрической энергией, газом и паром; кондиционирование воздуха» вычислялся как среднее арифметическое взвешенное среднегодовых уровней загрузки мощностей каждого из трех этих видов деятельности (57,3%, 60,5%, 66,3%) за 2024 г., где весами являются отношения среднегодовой стоимости основных фондов в среднегодовых ценах за 2023 г. каждого из этих видов деятельности (69852798, 63853961 и 51197597 млн. руб.) к сумме среднегодовых стоимостей в среднегодовых ценах по всем трем видам деятельности (184904356 млн. руб.). Подробная методика вычисления данного показателя на основе данных Росстата и Российского экономического барометра представлена в работе [Афанасьев, Пономарева, 2023].

2. Среднегодовая численность занятых в российском народном хозяйстве, по данным Росстата, выросла на 0,8% против предполагаемых авторами 0,7%. В среднем за 2024 г. численность рабочей силы в возрасте 15 лет и старше составила 76,1 млн. чел. против 76,0 в 2023 г. [Социально-экономическое положение России, 2024, с. 214]. Отметим, что в целях сопоставимости данных с 1990 г. авторы учитывают среднегодовой показатель занятости, исчисляемый по методике, действовавшей до 2016 г.
3. Цена на сырую нефть марки «Юралс» составила 59 долл./барр. в долларах США 2010 г. против спрогнозированного авторами значения 60 долл./барр. Этот показатель вычислялся как частное между номинальной среднегодовой ценой на нефть марки «Юралс» 67,85 долл./барр. и отношением среднегодовой номинальной цены на нефть марки «Брент» (80,70 долл./барр.) к среднегодовой цене на нефть марки «Брент» в долларах США 2010 г. (70,11 долл./барр.), вычисленным на основе данных Мирового банка¹. В свою очередь, среднегодовое значение номинальной мировой цены на нефть марки «Юралс» было вычислено авторами на основе данных Минэкономразвития России как среднее арифметическое ее среднемесячных значений. Среднемесячное значение цены за январь 2024 г. составило 64,14 долл./барр., за февраль – 68,33 долл./барр., за март – 70,34 долл./барр., за апрель – 74,98 долл./барр., за май – 67,37 долл./барр., за июнь – 69,58 долл./барр., за июль – 74,01 долл./барр., за август – 70,27 долл./барр., за сентябрь – 63,57 долл./барр., за октябрь – 64,72 долл./барр., за ноябрь – 63,48 долл./барр.², за декабрь – 63,44 долл./барр.³
4. Рост среднегодовой стоимости основных фондов в 2024 г. остается прежним – в размере 3%, поскольку данные по этому показателю формируются Росстатом к осени 2025 г.

С учетом приведенных выше уточненных величин аргументов функции (1) прогнозные на 2024 г. объемы ВВП России по всем обучающим выборкам, начинающимся с 1990 г. и оканчивающимися 2001–2023 гг. составили от 821 до 878 млрд. руб., а прогнозные на 2024 г. темпы роста ВВП – от 4,1% до 4,4% (рис. 1).

Если рассматривать досанкционный период 1990–2014 гг., то наименьшие средние ошибки ex-post прогноза до 2023 г. (4,1 – 4,2%) получаются по экзаменационным выборкам 2001–2023 гг., 2013–2023 гг. и 2014–2023 гг. на основе функции (1), МНК-оценки параметров которой вычислены по обучающим выборкам 1990–2000 гг., 1990–2012 гг. и 1990–2013 гг. соответственно (табл. 2, рис. 1). По этим трем выборкам прогнозные на 2014 г. значения ВВП России составили 837, 827 и 836 млрд. руб. (рис. 2 и 3), а прогнозные темпы его роста 4,2%, 4,2%, 4,1% соответственно. Согласно первой оценке Росстата, в 2024 г. фактический темп роста ВВП России составил 4,1% [Социально-экономическое положение России, 2024, с. 6], а объем ВВП в ценах 1990 г. – 837 млрд. руб.

Таким образом, в условиях усиливающегося санкционного давления, связанного с проведением специальной военной операции, в 2024 г. экономика России продемонстрировала достаточно высокие темпы экономического роста в 4,1% (против 3,6% в 2023 г.). Исследованная авторами макроэкономическая производственная функция (1) дала точные прогнозы на 2024 г. как ВВП России (836 – 837 млрд. руб.), так и темпов его роста (4,1% – 4,2%) (рис. 1–4). Основными факторами роста российской экономики в 2024 г. послужили основные фонды, уровень их загрузки в промышленности, мировая цена на нефть, в то время как уровень занятости оказал на него относительно слабое влияние. Можно заключить, что антироссийские санкции и военная поддержка Украины со стороны западных стран 2022–2024 гг. не только не дали ожидаемого их правительствами народнохозяйственного эффекта, но и поспособствовали ускорению темпов роста экономики России.

¹ <https://thedocs.worldbank.org/en/doc/5d903e848db1d1b83e0ec87f44e55570-0350012021/related/CMO-Historical-Data-Annual.xlsx>

² https://www.economy.gov.ru/search/?d=all&d_from=&d_to=&e=0&m=&q=%D0%BE%20%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B5%D0%B9%20%D1%86%D0%B5%D0%BD%D0%B5%20%D0%BD%D0%B0%20%D0%BD%D0%B5%D1%84%D1%82%D1%8C%20%D1%81%D0%BE%D1%80%D1%82%D0%B0%20%D1%8E%D1%80%D0%B0%D0%BB%D1%81&s=date&search=1&t=

³ https://economy.gov.ru/material/departments/d12/konyunktura_mirovyh_tovarnyh_rynkov/o_srednem_urovne_cen_nefti_sorta_yurals_za_dekabr_2024_goda.html

Таблица 1. Данные

Год t	V_t , тыс. чел	O_t , долл./бар.	Y_t , млрд. руб.	Z_t , млн. руб.	n_t , %
2024	69694	59,00	837*	3365905**	77
2023	69141	56,59	804	3267869	76
2022	67533	69,00	776	3179912	76
2021	67155	70,04	793	3081807	75
2020	65953	42,73	757	2976450	79
2019	67388	64,37	781	2853595	79
2018	68016	70,01	771	2762511	78
2017	68127	55,91	753	2730170	79
2016	68430	46,98	742	2696319	77
2015	68389	53,65	744	2673133	75
2014	67813	91,59	765	2644159	77
2013	67901	99,21	760	2581327	78
2012	67968	101,61	750	2499424	79
2011	67727	99,97	725	2416816	78
2010	67577	79,64	695	2350079	72
2009	67463	64,13	665	2292706	65
2008	68474	94,95	722	2229842	77
2007	68019	76,18	686	2169707	80
2006	67174	72,72	632	2119496	78
2005	66792	62,07	584	2074736	76
2004	66407	45,05	549	2040209	74
2003	65979	36,24	512	2015564	73
2002	65574	32,99	477	1993845	70
2001	64980	31,89	456	1976006	69
2000	64517	35,54	434	1962932	66
1999	63963	22,10	394	1953747	62
1998	63642	15,48	371	1953216	55
1997	64639	22,22	391	1967098	54
1996	65950	22,90	386	1983823	54
1995	66441	18,57	400	1995229	60
1994	68484	18,91	417	2014984	61
1993	70852	19,72	478	2030396	74
1992	72071	23,14	523	2009054	73
1991	73848	24,50	612	1957288	100
1990	75325	28,65	644	1871649	100

*Согласно первой оценке Росстата [Социально-экономическое положение России, 2024, с. 6].

** Ожидаемое авторами значение [Афанасьев, Пономарева, 2024].

Источники: [Афанасьев, Пономарева, 2024], Росстат, [Социально-экономическое положение России, 2024].

Таблица 2. Результаты МНК-исследования функции (1)

Годы	МНК-оценки параметров и t -статистики			R^2	DW
	α	γ	δ		
1990–2013	-7,133 (-24,5)	0,687 (6,779)	0,0037 (6,219)	0,953	1,136
1990–2012	-7,177 (-24,5)	0,700 (6,893)	0,0038 (6,338)	0,950	1,267
1990–2000	-7,352 (-28,7)	0,764 (8,19)	0,0028 (1,019)	0,921	1,803

Источник: (Афанасьев, Пономарева, 2024).

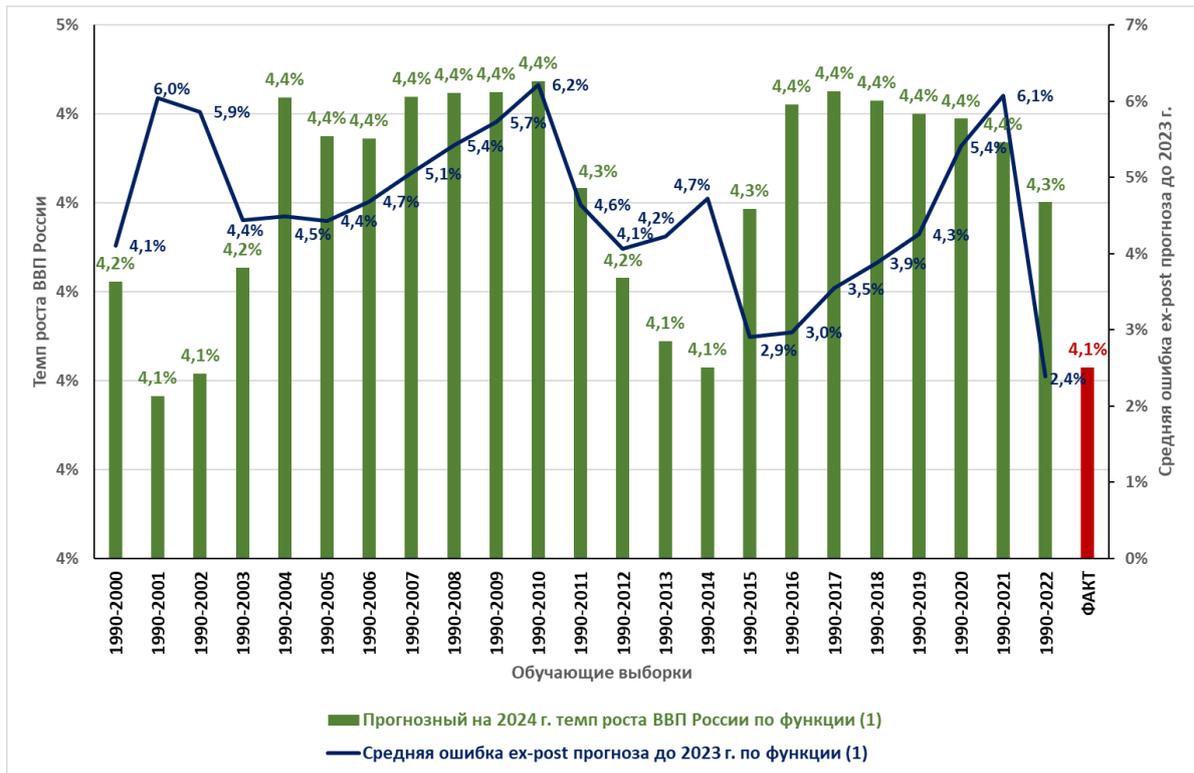


Рис. 1. Фактический за 2024 г., прогнозные на 2024 г. темпы роста ВВП России и средние ошибки ex-post прогноза до 2023 г. по функции (1), оцененной с 1990 г. по 2000–2022 гг.

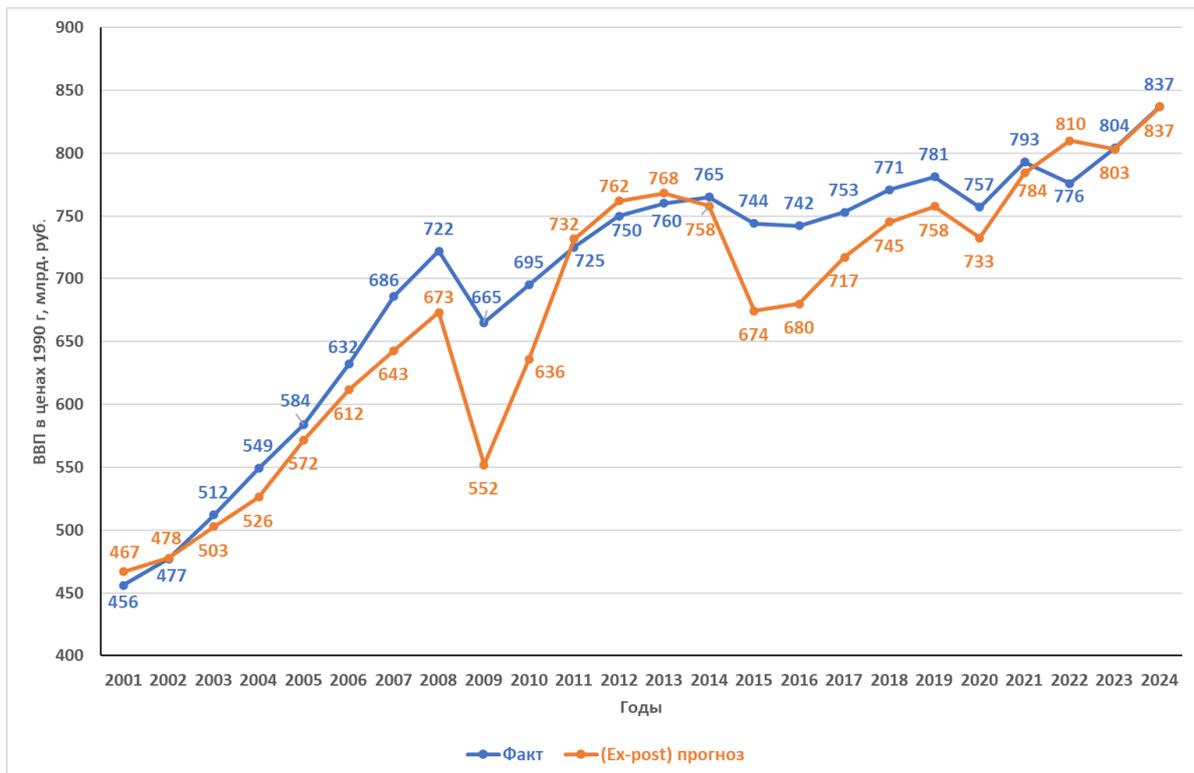


Рис. 2. Фактический и (ex-post) прогнозный на 2024 г. ВВП России по функции (1), оцененной в 1990–2000 гг.

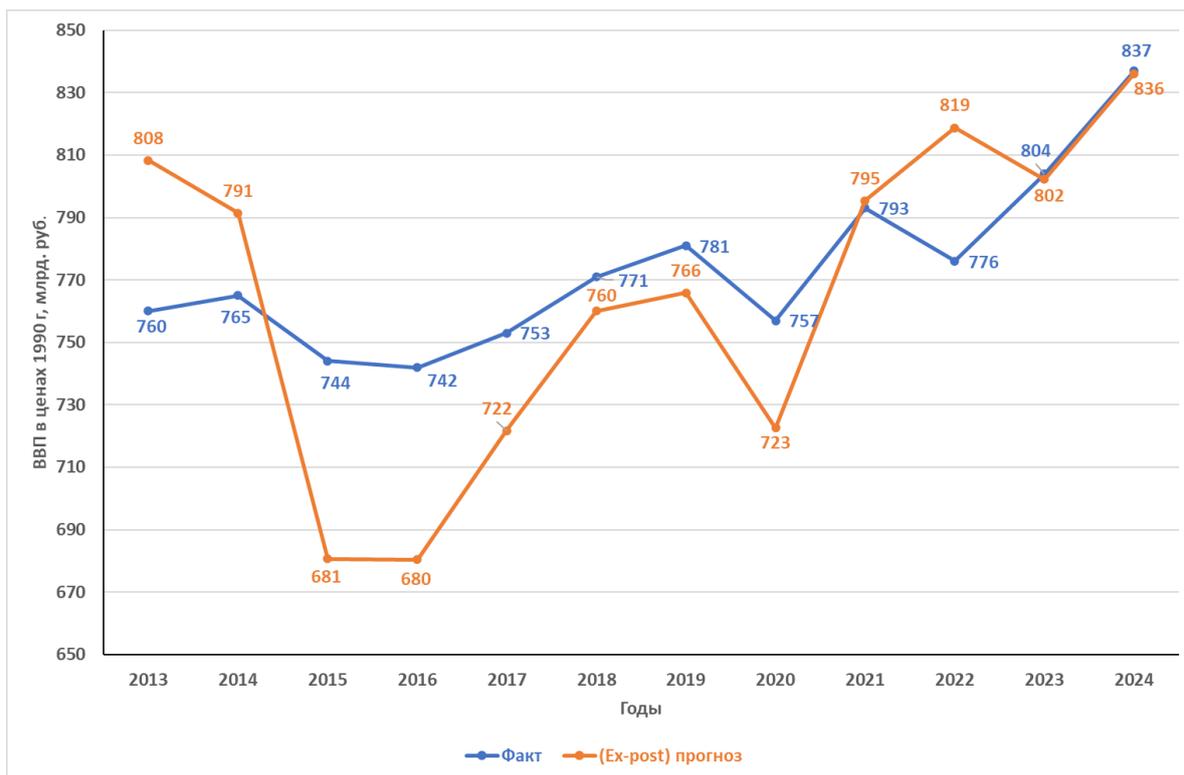


Рис. 3. Фактический и (ex-post) прогнозный на 2024 г. ВВП России по функции (1), оцененной в 1990–2012 гг.

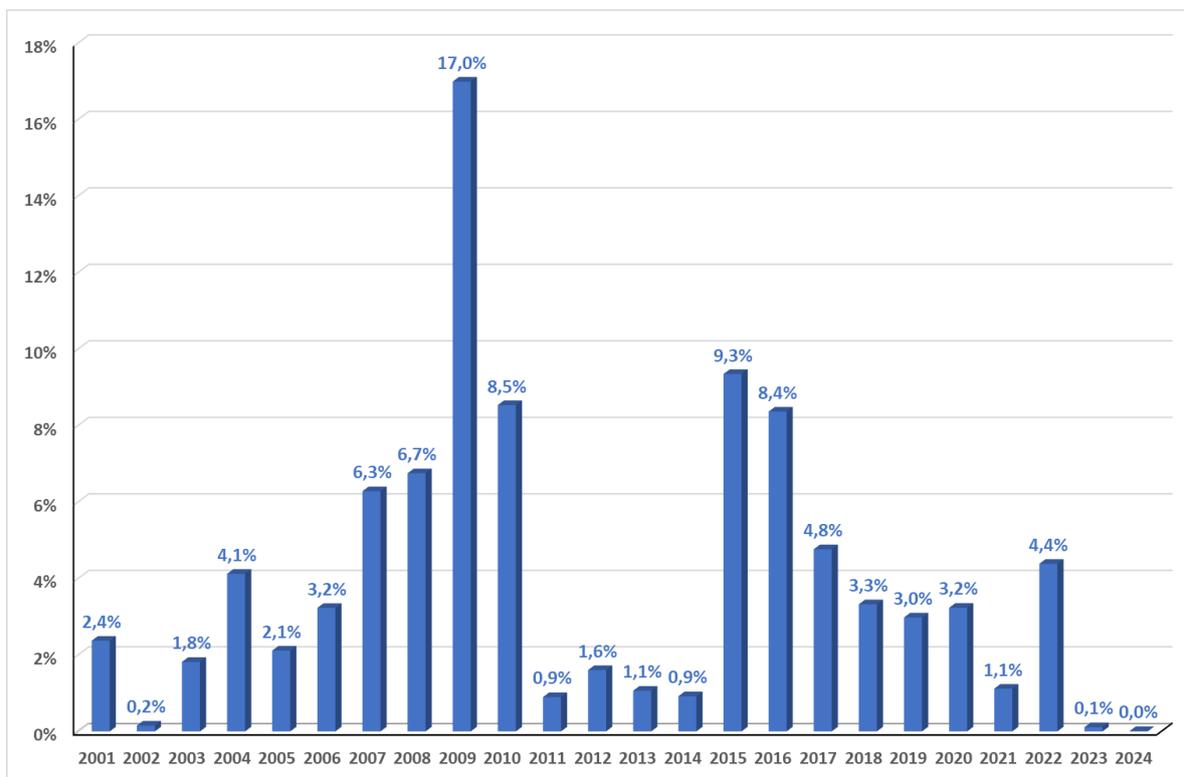


Рис. 4. (Ex-post) прогнозные на 2001–2024 гг. ошибки ВВП России по функции (1), оцененной в 1990–2000 гг.

Литература

1. Афанасьев А.А., Пономарева О.С. Эконометрический прогноз на 2024 год темпа роста народного хозяйства России в условиях нарастающего внешнего санкционного давления // Проблемы рыночной экономики. – 2024. – № 3. – С. 81-91.

2. Афанасьев А.А., Пономарева О.С. Эконометрическое прогнозирование на 2023 год ВВП России в условиях санкционных ценовых ограничений на экспорт нефти марки «Юралс» // Проблемы рыночной экономики. – 2023. – № 3. – С. 23-36.
3. Социально-экономическое положение России 2024 г. / Росстат. – М., 2024. – 356 с..

References in Cyrillics

1. Afanasiev A.A., Ponomareva O.S. Econometric forecast of the Russian economy growth for 2024 under increasing external sanctions pressure // Market economy problems. – 2024. – No. 3. – Pp. 81-91 (In Russian).
2. Afanasiev A.A., Ponomareva O.S. Econometric forecasting of Russian GDP for 2023 under the sanctioned price cap on the Urals crude oil export // Market economy problems. – 2023. – No. 3. – Pp. 23-36 (In Russian).
3. Socio-economic Situation in Russia 2024 / Federal State Statistics Service. – Moscow, 2024. – 356 p. (In Russian only)..

Афанасьев Антон Александрович, доктор экономических наук, доцент, ведущий научный сотрудник ЦЭМИ РАН, Москва; заведующий Кафедрой моделирования и системного анализа Факультета информационных технологий и анализа больших данных Финансового университета при Правительстве Российской Федерации, Москва.
(aanton@cemi.rssi.ru; aaafanasev@fa.ru)

Пonomарева Ольга Станиславна, старший научный сотрудник Центрального экономико-математического института РАН, Москва.
fondf@cemi.rssi.ru

Ключевые слова

исполнение, уточненный эконометрический прогноз, ВВП России, специальная военная операция России на Украине

Anton A. Afanasiev, Olga S. Ponomareva, Implementation of the revised econometric forecast of Russian GDP for 2024 under a special military operation

Keywords

Implementation of the revised econometric forecast of Russian GDP for 2024 under a special military operation

.DOI: 10.34706/DE-2025-01-07

JEL classification - Торговля: прогноз и моделирование, F40 - Макроэкономические аспекты международной торговли

Abstract

The authors present a revised econometric forecast for 2024 of the Russian GDP growth rate under the special military operation, based on the national economic production function, and the accuracy of its implementation. After Russian Federal State Statistics Service and the World Bank published actual data for 2024, it turned out that the values of such factors as the level of capacity use and the world price of Urals oil were slightly lower than expected by the authors. As a result, the values of the revised forecast GDP growth rates slightly decreased, amounting to 4.1–4.2% against the initially forecasted 5.4–5.5%, and thus almost coincided with the actual growth rate of 4.1%, according to the first estimate of Russian Federal State Statistics Service. The main factors of growth in 2024 were fixed assets, the level of their use in industry, and the world price of oil, while the employment level had a relatively weak impact on it. The studied production function can be successfully used for further forecasting of the dynamics of economic growth in Russia under increasing sanctions pressure from the collective West.

УДК: 332. 145

1.8. Использование нейросетей в маркетинговой деятельности строительной компании

Ратман Л.П., аспирант ГАУГН, г. Москва, Россия

Рассмотрено влияние искусственного интеллекта на сферу маркетинговой деятельности. Дано описание нейросетей и их применения в современном маркетинге. Представлены преимущества, которые дает интеграция ИИ и нейросетей с рекламными технологиями для анализа данных о потребностях клиентов, прогнозировании спроса на продукцию. Исследована возможность обучения нейронных сетей. Дана оценка скорости их развития и адаптации к исследуемым запросам.

Введение

Технологии искусственного интеллекта (ИИ) проникают во все сферы жизнедеятельности человека. Сфера маркетинговой деятельности не стала исключением. Маркетинг – один из ключевых элементов успешного развития любого бизнеса, в том числе строительной компании. С появлением технологий искусственного интеллекта исследования рынка и прогнозирование потребностей клиентов стали более точными и эффективными. Возможности искусственного интеллекта оперативно и точно проводить динамический анализ таких ключевых факторов, как потребительское поведение, уровень дохода, дифференциация цен и динамика рынка, стоимость владения продуктом для клиента, позволяют использовать его в разработке маркетинговых стратегий привлечения и удержания потребителей [1].

Внедрение ИИ в бизнес

Исследователи из Высшей школы экономики [предсказывают](#), что уже к 2025 г. ИИ оставит без работы значительную часть маркетологов. Для тех же, кто останется в профессии, работа с искусственным интеллектом станет обязательным навыком [2].

В основе современных технологий ИИ, которые трансформируют бизнес и общество, лежат машинное обучение, глубокое обучение и нейронные сети. Их использование позволяет автоматизировать процессы, улучшать взаимодействие с клиентами и находить новые способы повышения эффективности. В будущем эти технологии будут интегрироваться в бизнес-процессы, открывая новые возможности для компаний и организаций. [3]

Нейросети – это математические модели, построенные по образу и подобию работы нейронных сетей мозга, одно из направлений работы в сфере ИИ. Они способны обрабатывать большие объемы данных и выявлять закономерности, которые недоступны для человеческого разума.

Модели глубокого обучения работают на основе искусственных нейронных сетей, которые имитируют работу мозга человека. Сеть состоит из множества слоев, каждый из которых анализирует данные и передает результат следующему слою. Это позволяет машинам решать сложные задачи, такие как обработка изображений, распознавание голоса и перевод текста.

Глубокое обучение – подвид машинного обучения, который использует многослойные нейронные сети для анализа данных. Этот метод позволяет машине не просто обучаться на данных, но и делать более сложные выводы благодаря многослойной структуре обработки информации. Нейронные сети представляют собой ключевую технологию, которая лежит в основе как машинного обучения, так и глубокого обучения. Они состоят из множества узлов (нейронов), которые обрабатывают входные данные и принимают решения на основе обученных паттернов [3].

Применение нейросетей в маркетинге дает возможность значительно улучшить стратегию продвижения продукции и услуг компании, так как позволяет анализировать данные более точно и быстро, принимать обоснованные решения на основе информации, полученной из больших объемов данных. Чат-боты, созданные искусственным интеллектом, улучшают пользовательский опыт и становятся одним из самых мощных современных инструментов для профессионалов цифрового маркетинга [1].

В современном мире все больше компаний обращаются к использованию искусственного интеллекта с целью повышения эффективности своих маркетинговых решений. Например, интеграция ИИ и нейросетей с рекламными технологиями открывает перед бизнесом уникальные возможности для более эффективных и целенаправленных кампаний, аналитики и взаимодействия с заказчиками, позволяя осуществлять:

- аналитику данных;
- персонализацию и сегментацию;
- улучшение качества обслуживания;
- прогнозирование спроса и инвестиций;
- повышение качества обслуживания клиентов;
- оптимизацию цен и др. [4].

Нейросети активно применяются в маркетинге строительной отрасли, где важно не только поддерживать связь с заказчиками и привлекать новых клиентов, но и оптимизировать процессы строительства

и управления проектами. Использование нейросетей позволяет строительным компаниям автоматизировать многие процессы маркетинга, улучшить взаимодействие с клиентами и повысить конкурентоспособность на рынке.

Как упоминалось ранее, одним из ключевых преимуществ использования нейросетей в маркетинге строительной компании является возможность анализировать поведение потенциальных и существующих клиентов. На основе данных о предпочтениях, запросах и поведенческих паттернах пользователей нейросети могут создавать персонализированные маркетинговые кампании, которые будут более эффективными и привлекательными для аудитории. Кроме того, нейросети могут предсказывать спрос на определенные товары и услуги, что помогает компаниям оптимизировать свое предложение, управлять запасами и планировать производственные процессы. Это позволяет сэкономить ресурсы и повысить эффективность всей цепочки поставок.

Еще одним важным достоинством использования нейросетей в маркетинге строительных компаний является возможность прогнозировать рыночные тенденции и анализировать конкурентную среду. На основе данных о конкурентах и рыночной динамике нейросети могут помочь компаниям выстроить успешную стратегию маркетинга, адаптировать свои предложения к изменяющимся условиям и выделиться на фоне других участников рынка.

Разберем, как именно нейросети помогают в конкурентном анализе.

Сбор и обработка данных об игроках рынка

Ручной сбор информации о конкурентах — трудоемкая и устаревшая практика. Нейросети позволяют автоматизировать этот процесс, анализируя сайты компаний, социальные сети, форумы, новостные публикации и базы данных, осуществляя:

- парсинг данных — алгоритмы анализируют веб-страницы конкурентов, выявляя ключевые параметры: ассортимент, цены, акции, отзывы клиентов и даже изменения в персонале;
- анализ текстов — технологии обработки естественного языка (NLP) позволяют нейросетям извлекать полезную информацию из статей, пресс-релизов и отчетов конкурентов;
- кластеризацию данных — ИИ группирует компании по сходным характеристикам, выявляя скрытые закономерности в их бизнес-моделях.

Автоматический мониторинг цен и трендов

Ценообразование и тренды в отрасли меняются стремительно. Компании, которые полагаются на ручной мониторинг, рискуют отстать от конкурентов. Нейросети способны анализировать сотни тысяч ценовых предложений и рыночных факторов в реальном времени.

Ключевые возможности:

- слежение за изменением цен — ИИ анализирует прайс-листы конкурентов и автоматически корректирует стратегию ценообразования;
- прогнозирование трендов — на основе исторических данных и текущих изменений нейросети предсказывают будущие тенденции в отрасли;
- выявление аномалий — системы машинного обучения помогают обнаружить резкие изменения в рынке, которые могут сигнализировать о кризисах или новых возможностях.

Инструменты для конкурентного анализа с ИИ

Современный рынок предлагает множество решений на основе ИИ, упрощающих анализ конкурентов. Вот некоторые из наиболее эффективных:

- Crayon — автоматизирует мониторинг конкурентов, собирая данные из тысяч источников.
- Kompyte — анализирует изменения в маркетинговых стратегиях конкурентов и предлагает рекомендации.
- Price2Spy — отслеживает динамику цен и помогает оптимизировать стратегию ценообразования.
- Brandwatch — анализирует упоминания брендов и конкурентов в соцсетях и СМИ.
- SimilarWeb — предоставляет аналитику по веб-трафику конкурентов, включая источники посещений и поведение пользователей [5].

Обучение нейросетей

Правильное использование нейросетей при выборе эффективных маркетинговых решений для строительной компании позволяет не только оптимизировать маркетинговые процессы, но и повысить конкурентоспособность компании в целом, значительно улучшить бизнес-процессы.

Наиболее важные задачи, которые способна решать обычная нейросеть — это классификация, распознавание и предсказание. По сути, нейросети могут самостоятельно развиваться, обучаться и «строить» свой опыт, исходя из своих же ошибок. Способность к обучению является наиважнейшим свойством человеческого мозга. В отношении нейросетей процесс обучения рассматривается как настройка архитектуры сети для наиболее эффективного выполнения определенной задачи. В большинстве случаев нейронная сеть настраивает свои веса связей по уже имеющейся обучающей выборке, функционирование сети будет улучшаться по мере итеративной настройки весовых коэффициентов [6].

Одной из целей использования нейросетей в маркетинге строительной компании является анализ данных о потребностях клиентов. Нейросети способны анализировать большие объемы информации,

такие как отзывы клиентов, социальные сети, статистические данные, прогнозы экономистов, и предлагать оптимальную стратегию маркетинговой деятельности с учетом потребностей и предпочтений целевой аудитории.

Еще одна задача – применение нейросетей для прогнозирования спроса на продукцию и услуги строительной компании. На основе исторических данных и текущих тенденций нейросети способны предсказать будущий спрос на продукцию компании и помочь оптимизировать производственные процессы. Также нейросети могут использоваться для улучшения работы отдела маркетинга строительной компании, а именно: автоматизировать процессы анализа рынка, поиска потенциальных клиентов и разработки индивидуальных маркетинговых стратегий.

Таким образом, использование нейросетей в маркетинговой деятельности строительной компании может способствовать повышению эффективности продвижения продукции и услуг компании на рынке, привлечению клиентов, увеличению прибыли, укреплению позиций на рынке.

В качестве примера изучения способности нейросетей к обучению и скорости их развития мною был проведен анализ ключевых запросов и ответов на них нейросети Яндекс с разницей примерно в один месяц.

Получены следующие результаты:

Вопрос в поисковой строке: «Сколько стоит построить одноэтажный дом из газоблока?»

Ответ нейросети от 8 января 2025 г.: «Стоимость одноэтажного дома из газоблока размером 100 м² без отделки может сильно различаться в зависимости от местоположения, качества материалов, сложности проекта и других факторов. Однако в среднем цена за строительство такого дома может начинаться от 1 500 000 руб. Точную цену лучше уточнить у специализированных компаний, занимающихся строительством загородных домов из газоблока.»

Приведенная стоимость оказалась занижена примерно в 2 раза по сравнению с тем, что в настоящее время предлагается на рынке. Ответ нельзя считать корректным, также недопустимо опираться на подобный расчет при формировании цен.

12 февраля 2025 г. ответ нейросети Яндекс был следующим:

«Стоимость строительства одноэтажного дома из газоблока зависит от многих факторов, таких как размер дома, планировка, используемые материалы и технологии. Примерная стоимость строительства дома размером 8,6 на 16,6 метров с террасой и крыльцом составляет около 7 266 876 руб.»

Очевидно, что за месяц ответ качественно улучшился, приведены конкретный пример и стоимость, поэтому данный результат можно использовать в анализе рынка малоэтажного строительства.

На запрос от 9 января 2025 г.: «Какие проекты одноэтажных домов из газоблока площадью 100 м² сейчас популярны?», нейросеть выдала следующую информацию:

«1. Проект одноэтажного дома из газоблока площадью 100 м² с двумя спальнями, гостиной, кухней, ванной комнатой и террасой.

2. Дом-коттедж из газоблока площадью 100 м² с открытой планировкой: гостиная, кухня-столовая, спальня, ванная комната и терраса.

3. Уютный одноэтажный дом из газоблока площадью 100 м² с тремя спальнями, гостиной, кухней, ванной комнатой и верандой.

4. Проект дома из газоблока площадью 100 м² с двумя спальнями, кухней-гостиной, ванной комнатой, просторной террасой и гаражом.

5. Современный одноэтажный дом из газоблока площадью 100 м² с двумя спальнями, гостиной, кухней-столовой, ванной комнатой, сауной и бассейном.»

Полученную информацию можно использовать при выборе проектов для рекламных компаний, поскольку в данном случае нейросеть, используя объем поисковых запросов целевой аудитории, сформировала определенные критерии популярности проектов домов и правильно ранжировала их по порядку.

Спустя месяц на тот же запрос нейросеть ответила следующее (ответ от 12 февраля 2025 г.):

«Вот несколько популярных проектов одноэтажных домов из газоблока площадью 100 м²:

1. Проект одноэтажного дома AS-2236 со стеклянным фасадом и четырьмя комнатами.

2. Проект одноэтажного загородного дома AS-2071 с крыльцом.

3. Проект одноэтажного дачного дома AS-2054 из газобетона.

4. Проект одноэтажного дома AS-2573 с террасой и крыльцом.

5. Проект одноэтажного дома AS-2742 в классическом стиле.

6. Проект одноэтажного дома AS-2476 с крыльцом и террасой.

7. Проект одноэтажного дома AS-2601-2 с крыльцом и террасой.

8. Проект бани из пенобетона AS-2000-6 с террасой и комнатой отдыха.

9. Проект одноэтажного дома AS-2319 с подвалом и эркером.

10. Проект одноэтажного дома AS-2205 с сауной, террасой и двускатной крышей.

11. Проект одноэтажного квадратного дома AS-2214 с чердаком.

12. Проект одноэтажного дачного домика AS-2142-2 из газобетона 6 на 6 метров.»

В данном случае мы получили примеры конкретных проектов, доступных в интернете, что также можно использовать в интересах компании при исследовании рынка.

В то же время с запросом от 9 января 2025 г., в котором были указаны более «размытые» параметры, а именно: «Какие дома из газоблока популярны в данный момент?», нейросеть откровенно не справилась и выдала ошибочный ответ, перепутав дома из керамического блока с домами из газосиликатного блока:

«В настоящее время наиболее популярны для строительства дома из газоблока различных марок, таких как Porotherm, ThermoBrick, Roca, Penoplex и др.

Наибольший спрос отмечается на дома из газоблока Porotherm. Они отличаются высоким качеством, хорошей теплоизоляцией, надежностью и долговечностью. Кроме того, благодаря простоте и скорости монтажа такие дома строятся быстрее, чем традиционные кирпичные дома.

Также популярными являются дома из газоблока ThermoBrick и Roca. Они также обладают хорошими теплоизоляционными свойствами и низкой теплопроводностью, что позволяет сэкономить на отоплении и кондиционировании воздуха.

В целом, дома из газоблока пользуются спросом благодаря своей экологичности, энергоэффективности, прочности и скорости монтажа.»

Удовлетворительный ответ, касающийся архитектурных стилей, последовал 12 февраля 2025 г.:

«В данный момент популярны дома из газоблока в следующих архитектурных стилях:

«хай-тек» или «минимализм»;

скандинавский стиль;

французский стиль «шале»;

«фахверк»;

стиль Райта (также известный как «стиль прерий»).»

Несмотря на то, что самыми популярными являются классические дома, полученная информация может быть полезной при выборе вариантов домов для рекламных компаний.

Выводы

Проведенный анализ позволяет сделать вывод, что, несомненно, за нейросетями будущее, и правильная работа с ними может быть успешной, но пока объем их «знаний» недостаточен, как и не вполне развита способность анализировать информацию. Поэтому в настоящее время грамотно использовать нейросети для формирования маркетинговой стратегии строительной компании может только квалифицированный специалист, разбирающийся как в строительстве, так и в маркетинге. Однако с учетом тенденций и скорости развития нейросетей их будущее выглядит оптимистично. Всего за месяц качество ответов нейросети на одни и те же запросы значительно повысилось. Очевидно, что уже в ближайшее время они станут основным источником информации, причем проанализированной, ранжированной по порядку и пригодной для использования даже без глубокого анализа и переработки.

Литература

1. Агаметов И.Э. Применение искусственного интеллекта и машинного обучения в практике цифрового маркетинга / И. Э. Агаметов, Е. А. Вечкинзова // Креативная экономика. – 2023. – Т. 17, № 8. – С. 2745-2760. [Электронный ресурс]. – URL: <https://elibrary.ru/item.asp?id=54383738> – DOI 10.18334/ce.17.8.118923 (дата обращения: 12.02.2025).
2. Искусственный интеллект в маркетинге: большой обзор / Skillbox Media [Электронный ресурс]. – URL: <https://skillbox.ru/media/marketing/iskusstvennyy-intellekt-v-marketinge-gde-ego-ispolzuyut-i-kak-vnedrit-uzhe-zavtra/?ysclid=m72y2hun62608179778> (дата обращения: 12.02.2025).
3. Основные технологии ИИ: машинное обучение, глубокое обучение и нейронные сети [Электронный ресурс]. – URL: <https://neiroseti.ai/tpost/1h8ksbsbo1-osnovnie-tehnologii-ii-mashinnoe-obuchen>(дата обращения: 12.02.2025).
4. Бурханов Т.Р. Роль нейросетей в маркетинговом продвижении компании на B2C рынке / Бурханов Т.Р., Кошель В.А. // Практический маркетинг, № 12. – 2023. [Электронный ресурс]. – URL: <https://cyberleninka.ru/article/n/rol-neyrosetey-v-marketingovom-prodvizhenii-kompanii-na-b2c-rynke/viewer>(дата обращения: 12.02.2025).
5. Рогова И. Практическое применение: как нейросети могут помочь в анализе конкурентов – Маркетинг на vc.ru [Электронный ресурс]. – URL: <https://vc.ru/marketing/1804287-prakticheskoe-primeneniye-kak-neiroseti-mogut-pomoch-v-analize-konkurentov> (дата обращения: 12.02.2025).
6. Акулин Е. В. Обучение нейронных сетей / Е. В. Акулин, Л. Е. Свиридова. – Текст : непосредственный // Исследования молодых ученых : материалы XXVIII Междунар. науч. конф. (г. Казань, декабрь 2021 г.). – Казань: Молодой ученый, 2021. – С. 6-10. [Электронный ресурс]. – URL: <https://moluch.ru/conf/stud/archive/410/16843/> (дата обращения: 14.02.2025).

Дополнительные источники

1. Паспорт национального проекта «Национальная программа «Цифровая экономика Российской Федерации», утв. президиумом Совета при Президенте РФ по стратегическому развитию и национальным проектам, протокол от 04.06.2019 № 7 [Электронный ресурс]. – URL: https://d-russia.ru/wp-content/uploads/2019/02/pasport_natsprogrammy_ifr_economika_oficialno.pdf

2. Цифровая экономика: 2025: краткий стат. сб. / В. Л. Абашкин, Г. И. Абдрахманова, К. О. Вишневецкий, Л. М. Гохберг и др.; Нац. исслед. ун-т «Высшая школа экономики». – М.: ИСИЭЗ ВШЭ, 2025 – 120 с. [Электронный ресурс]. – URL: <https://issek.hse.ru/mirror/pubs/share/995751983.pdf?ysclid=m74xoylcrj355472022>

References in Cyrillics

1. Movchan A. E`konomika voennogo vremeni. Kak pandemiya 2020 goda izmenit mir. [E`lektronny`j resurs]. – URL: <https://www.forbes.ru/biznes/395899-ekonomika-voennogo-vremeni-kak-pandemiya-2020-goda-izmenit-mir> (data obrashheniya: 28.11.2025).
2. Badusheva V.D., Palagin A.A. Razvitie otrasli stroitel'stva pod vliyaniem COVID 19// Vestnik Akademii znaniy № 39(4). – 2020. – S. 82-84. [E`lektronny`j resurs]. – URL: <https://cyberleninka.ru/article/n/razvitie-otrasli-stroitelstva-pod-vliyaniem-covid-19/viewer> (data obrashheniya: 20.01.2024).
3. Stroitel'naya otrasl' v period pandemii: obzor delovogo klimata ot e`kspertov VShE`. [E`lektronny`j resurs]. – URL: <https://erzrf.ru/publikacii/stroitel'naya-otrasl-v-period-pandemii-obzor-delovogo-klimata-ot-ekspertov-vshe?ysclid=itd2yv6rh1182259395> (data obrashheniya: 20.01.2024).
4. Kak menyalsya stroitel'ny`j ry`nok na fone pandemii// Stroitel'naya gazeta – [E`lektronny`j resurs]. – URL: <https://stroygaz-ru.turbopages.org/turbo/stroygaz.ru/s/expert/construction/kak-menyalsya-stroitelnyy-rynok-na-fone-pandemii/> (data obrashheniya: 20.01.2024).
5. Faktor virusa. [E`lektronny`j resurs]. – URL: <https://zsrf.ru/blogpost/52/faktor-virusa> (data obrashheniya: 25.11.2024).
6. Xashimova D. P. Ispol'zovanie instrumentov cifrovogo marketinga dlya povыsheniya e`ffektivnosti torgovы`x operacij // «E`konomika i socium», №4(95). – 2022. – [E`lektronny`j re-surs]. – URL: https://www.iupr.ru/ files/ugd/b06fdc_331a6276689f4029b6acf35e697b79cf.pdf?index=true

Ключевые слова

Искусственный интеллект, нейронные сети, глубокое обучение, маркетинг, аналитика данных, прогнозирование спроса, оптимизация деятельности строительной компании.

Ратман Леонид Панаётович, аспирант,

ФГБОУ ВО ГАУГН

(leonidrat@yandex.ru)

ORCID: 0000-0003-2878-6255

Leonid Ratman. The impact of the global situation on the market of construction services and materials in Moscow and the Moscow region on the example of the COVID-19 pandemic

Keywords

Artificial intelligence, neural networks, deep learning, marketing, data analytics, demand forecasting, optimization of the construction company..

DOI: 10.34706/DE-2025-01-08

JEL classification C8 Методология сбора и оценки данных;

Abstract

The impact of artificial intelligence on the sphere of marketing activities is considered. A description of neural networks and their application in modern marketing is given. The advantages of integrating AI and neural networks with advertising technologies for analyzing data on customer needs and forecasting product demand are presented. The possibility of training neural networks has been investigated. The rate of their development and adaptation to the investigated requests is estimated.

УДК: 336. 22

1.9. Цифровое моделирование НДС и налога с оборота

Грачёв И.Д., Ларин С.Н., Ноакк Н.В.
ЦЭМИ РАН, Москва, Россия

Аннотация: *Изменения, происходящие в мировой экономической системе, поставили под сомнение всю аксиоматику типа Вашингтонского консенсуса и актуализировали научные исследования, в частности, по налоговым и таможенным системам. Эти проблемы важны и для нашей страны, но, к сожалению, подходы к их решению носят в основном вербальный характер и далеки от универсальности. В настоящей работе выполнено компаративное цифровое моделирование сравнительной эффективности налога с оборота и налога на добавленную стоимость для диаметрально различных по ресурсообеспеченности стран. Одна из них обеспечена ресурсами с хорошим запасом на будущее, но ее экономика сосредоточена на низких переделах. У второй полностью отсутствуют природные ресурсы, но есть технологическое превосходство, выражающееся в доминировании высоких переделов. На цифровых моделях показано, что при всех реалистичных уровнях технологического превосходства и уровнях налогообложения, а также полном отсутствии таможенных пошлин, выбор оптимального налога для страны с высокой ресурсообеспеченностью и высокотехнологичной страны диаметрально различен. Полученные в ходе цифрового моделирования результаты дополнительно, наряду с традициями и историей, аргументируют правомерность гипотезы об обоснованном отказе США от НДС. Сегодня США, как и Россия, относятся к ресурсообеспеченным странам, поскольку три из четырех первых позиций в её экспорте занимают нефть, продукты ее переработки и газ. На основе результатов цифрового моделирования сделан вывод о том, что в настоящее время у России имеется ещё больше оснований от отказа от НДС, чем у США.*

Введение

В практике последнего десятилетия, особенно после прихода к власти в США Трампа, актуализировалась научная дискуссия об эффективности расширенных налоговых систем, включающих налоги, пошлины, социальные сборы. Эта проблематика тем более актуальна для России, налоговая система которой «оптимизирована» под существенным влиянием сформированной в рамках Вашингтонского консенсуса аксиоматики (типа: свободная торговля – это всегда хорошо, а таможенные пошлины (тарифы) – это всегда плохо). Между тем эта аксиоматика уже перестала быть базовой в самом Вашингтоне.

Основной проблемой российской налоговой системы является противоречивое толкование и частые поправки налогового законодательства. Самыми актуальными для России в рамках этой проблематики являются:

1) так называемый налоговый маневр с заменой таможенных пошлин внутренними налогами, что с нашей точки зрения (и сегодня её продвигает Президент США Трамп) эквивалентно переносу налоговой нагрузки с граждан других стран на граждан своей страны для товаров, от которых мировая экономика не может отказаться, а именно таковыми являются: газ, нефть и продукты ее переработки;

2) использование налога на добавленную стоимость (НДС), которого нет в США, в качестве главного бюджетобразующего налога.

Ниже будет представлен обзор источников литературы, авторы которых аргументируют основные положения в пользу НДС и налога с оборота.

Обзор литературы

Налоги и сборы являются основой налоговой системы, через которую государство реализует свои функции по управлению (регулированию, распределению и перераспределению) финансовыми потоками и экономикой страны. С некоторым упрощением основным назначением налогов является наполнение государственного бюджета.

Основной проблемой российской налоговой системы является противоречивое толкование норм и практики налогообложения, а также введение многочисленных поправок в налоговое законодательство. Налоговый Кодекс РФ лидирует по числу ежегодно вносимых в него поправок среди федеральных законодательных актов. Кроме корректировки норм по конкретным налогам и сборам, это связано и с необходимостью постоянного совершенствования налоговой системы. Вместе с тем, нестабильность, хаотичный характер изменений, отсутствие системности в налоговом законодательстве усложняют практику его применения и сдерживают развитие экономики.

Налог на добавленную стоимость.

Дискуссия о недостатках НДС в сравнении с другими налогами идёт давно. С нашей точки зрения у этой дискуссии есть два основных недостатка. Во-первых, она в основном вербальная, и, во-вторых, явно или неявно она предполагает возможность универсальной оптимальности. В настоящее время активно обсуждается такой аспект трансформации налоговой системы России, как отмена налога на добавленную стоимость и введение вместо него налога с оборота (Минаева, Аношина, 2019).

Впервые НДС был введен во Франции в 1954 году по предложению руководителя дирекции по налогам и сборам Министерства экономики, финансов и промышленности Франции Мориса Лоре. В настоящее время он применяется более чем в 137 странах мира. В Великобритании он называется VAT (базовая ставка – 20%), в Германии – MwSt (19%), во Франции – TVA (19,6%). Наиболее высокие ставки по НДС в Венгрии – 27 %, а также в Швеции, Дании и Норвегии – по 25%. НДС не применяется в США, где пошлины намного меньше, чем отчисления по НДС в крупнейших странах мира, а также в ряде арабских стран (Кувейт, Катар, Оман, Бахрейн), обладающих большими запасами нефти (VisaSam, 2025).

В России с 2025 года действует ставка НДС в 0%, 10% или 20% в зависимости от перечня реализуемых услуг или товаров. При упрощенной системе налогообложения (УСН) действуют ставки 5% при доходе от 60 млн до 250 млн рублей и 7% при доходе от 250 млн до 450 млн рублей. В составе общего объема нефтегазовых доходов за 2019-2023 годы доля НДС составляет – 39,9% (в том числе НДС внутренний – 24,7%, НДС ввозной – 15,2%). Доля НДПИ (включая нефтегазовую часть) – 33,3%. Доля таможенных пошлин и сборов составила 8,4%, доля налога на прибыль – 6,6%, доля акцизов (без учета акциза на нефтяное сырье, направленное на переработку) – 4,2%. На структуру доходов существенное влияние оказывает возвратный акциз на нефтяное сырье, направленное на переработку, доля которого составила (-)10,0% (см. рис.1 (Исполнение федерального бюджета ..., 2023)).

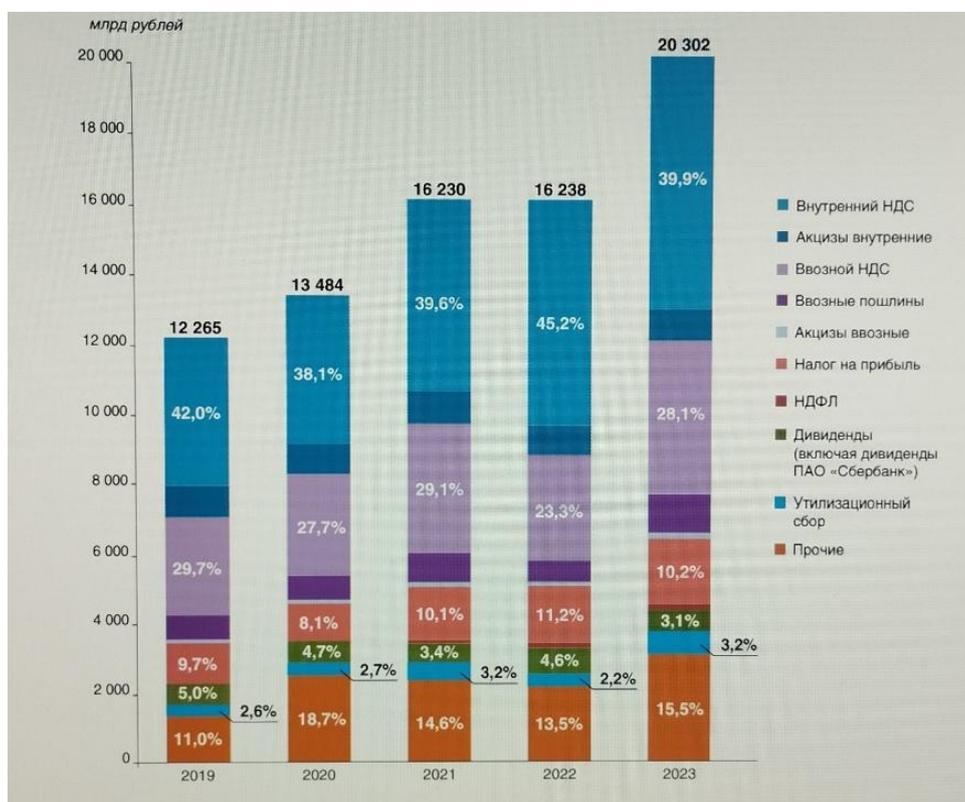


Рис.1. Доля НДС в структуре нефтегазовых доходов федерального бюджета в 2019÷2023 годах.

НДС относится к косвенным налогам, а его исчисление производится продавцом при реализации продукции покупателю. К добавленной стоимости относится та часть выручки, которая составляет разницу между прибылью от реализации и издержками на производство или закупку товара. Принцип применения налога везде одинаков - он рассчитывается исходя из цены товара или услуги с последующим вычетом из него налога, выставленного поставщиками. Поскольку сумма НДС заложена в цену товара, то его платит конечный потребитель. Однако перечисляет средства в бюджет и отчитывается перед ФНС продавец.

К преимуществам НДС, по мнению А.А. Бухаровой (Бухарова, 2017), следует отнести возможность его минимизации путем вычета входящего НДС для производителей, осуществляющих его перечисление. Вместе с тем, практика применения механизма исчисления НДС приводит к росту цен на продукты

и услуги в связи с ростом затрат на их производство и транспортировку. Следствием этого становится сокращение спроса и повышение неудовлетворенности потребителей.

В своей работе Ш.А. Магомадов отмечает одновременную простоту понимания и относительную сложность расчета и уплаты НДС (Магомадов, Джанаралиев, 2019). Поскольку косвенный налог, в отличие от прямого, оплачивает конечный покупатель товара, то для предпринимателя НДС не выглядит обременительным. Однако, учитывая действие нескольких ставок и многочисленных правил его расчета, можно говорить о наличии определенных сложностей со своевременным начислением и выплатой НДС.

Одной из проблем использования НДС является сложность правильного расчета налоговой базы для начисления (Буланова, 2021). Ошибки при ее определении ведут к неправильному расчету суммы и впоследствии к некорректной уплате НДС. Это рассматривается как налоговое правонарушение и может стать основанием для привлечения к уголовной или административной ответственности.

По мнению В.Г. Панскова (Пансков, 2020), высокие ставки НДС могут свидетельствовать о наличии проблем в развитии экономики отдельных стран.

К недостаткам НДС в работе (Кирееенко, Содномова, 2020) отнесены следующие: преобладает негативное влияние на денежные потоки плательщиков; сложность расчета ведет к росту издержек его взимания у государства и налогоплательщиков; низкое соотношение фактических доходов от НДС к потенциально возможному; отсутствие возможности предотвратить отдельные виды мошенничества с налогом. Авторы проанализировали альтернативные механизмы сбора НДС (механизм обратной оплаты, сплит-платежные механизмы, метод «единого окна»), которые эффективно снижают уклонение от его уплаты, возникающее при завышении вычетов по НДС. Однако они не могут использоваться для борьбы с сокрытием доходов. Кроме того, их внедрение влечет за собой рост издержек по налоговому администрированию, поскольку приводит к увеличению объема отчетности и появлению новых обязанностей для налогоплательщиков по уплате НДС.

Следует уточнить, что при выполнении обязательств по НДС возникает ряд рисков, которые при определенных обстоятельствах могут обусловить утрату его самых весомых преимуществ. могут обостриться и стать настоящим препятствием для результативности системы НДС. К таким обстоятельствам можно отнести:

- кризис платежеспособности, низкая конкурентоспособность товаров и услуг, затрудняющие их реализацию и возмещение НДС в их стоимости;
- сложность для производителей с пополнением оборотных средств в связи с высокими ставками по кредитам;
- открытость национальной экономики и высокая экспортная доля валового внутреннего продукта, что приводит к значительным требованиям к возмещению НДС предприятиям-экспортерам и снижению его эффективности с точки зрения налоговых сборов;
- непрозрачность реализации товаров и услуг, низкая правовая и налоговая грамотность увеличивают риски извлечения возмещаемого НДС вне правового поля, способствуют росту налоговых обязательств и уклонению от уплаты (Скобликов, 2014).

Наличие указанных рисков позволяет усомниться в эффективности НДС как одного из значимых в налоговой системе и выступает дополнительным аргументом в пользу возврата к введению налога с оборота.

Налог с оборота.

Очевидным единством НДС и налога с оборота является их многоэтапный характер. Он выражается в том, что взимание этих налогов происходит на каждом этапе движения товаров и услуг от производителя к конкретному потребителю. При начислении налога с оборота возникает кумулятивный эффект, который вызван включением в налоговую базу стоимости товаров и услуг, добавленной как на текущем, так и на предшествующих этапах, которые ранее уже облагались налогом (Скобликов, 2014). Поэтому для конечного покупателя налог с оборота может стать непомерно большим и привести к отказу от покупки.

Помимо этого, использование налога с оборота на практике приводит к иррациональному распределению налоговой нагрузки между производителями. Те из них, которые находятся на начальных этапах (добыча сырья/топливо, выработка электроэнергии) производства могут значительно минимизировать эту нагрузку и, наоборот, те производители, которые заняты в сфере конечной реализации (обрабатывающая промышленность, торговля и др.) вынуждены нести максимальную налоговую нагрузку (Бачурин, 2016).

Налог с оборота актуализирует вертикальную концентрацию производства, поскольку при слиянии сопряженных в технологической цепочке компаний отпадает необходимость уплаты налога с оборота на всех промежуточных этапах движения товаров и услуг в рамках их деятельности (Чехов, 2021).

На практике одной из сильных сторон налога с оборота по сравнению с НДС является простота определения налоговой базы и администрирования. Также к числу достоинств введения единого оборотного налога можно отнести:

- оптимизацию системы налогообложения за счет замены одним налогом четырех других – НДС, налога на прибыль, страховых взносов и НДФЛ;
- реализацию принципа равенства налогообложения;

- устранение налоговой дискриминации;
- максимальную простоту налогового администрирования;
- повышение инвестиционной активности;
- гармонизацию отраслевой структуры экономики.

Упрощению уплаты налога с оборота будет способствовать безакцептное изъятие суммы налога при каждом поступлении денежных средств на расчетный счет или при инкассации наличных денег в ритейле (Шамаев, 2024). Однако безакцептное взимание налога с оборота негативно скажется на малом и среднем бизнесе и заставит его: сократить число различных смежников, поставщиков; перестроить кооперационные и производственные связи, оптимизировать их; укрупнять заказы, поставки, объемы производства. Современная практика показывает, что для малого и среднего бизнеса проще применять УСН, ЕНВД, патентную систему и другие налоговые режимы, чего нельзя будет сказать о едином налоге с оборота. К тому же оборот не является показателем прибыльности, и даже при низкой ставке (6,2%) сумма налога с оборота может оказаться больше, чем сумма НДС (20%).

Упрощенный сбор налога с оборота могут обеспечить современные банковские цифровые технологии, включая электронные платежные системы. Они не только снижают стоимость сбора налогов, но и создают потенциал для расширения налоговой базы (например, путем улучшения идентификации и мониторинга налогоплательщиков и облегчения для налогоплательщиков соблюдения таких средств, как использование мобильных технологий).

По оценкам McKinsey & Company, цифровизация процессов государственных платежей (как доходов, так и расходов) может обеспечить экономию не менее 1% ВВП. Кроме того, на основе глобальной платёжной карты McKinsey, которая охватывает более 25 платёжных продуктов в 48 странах и более 90% мирового ВВП, было установлено, что в 2023 году мировая индустрия платежей обработала 3,4 триллиона транзакций на сумму 1,8 квадриллиона долларов, а выручка составила 2,4 триллиона долларов. С 2018 по 2023 год выручка ежегодно росла на 7%, но в ближайшие пять лет рост замедлится до 5% в год. К концу 2028 года выручка увеличится ещё на 700 миллиардов долларов и составит 3,1 триллиона долларов (см. рис. 2 (The 2024 McKinsey Global Payments Report)).

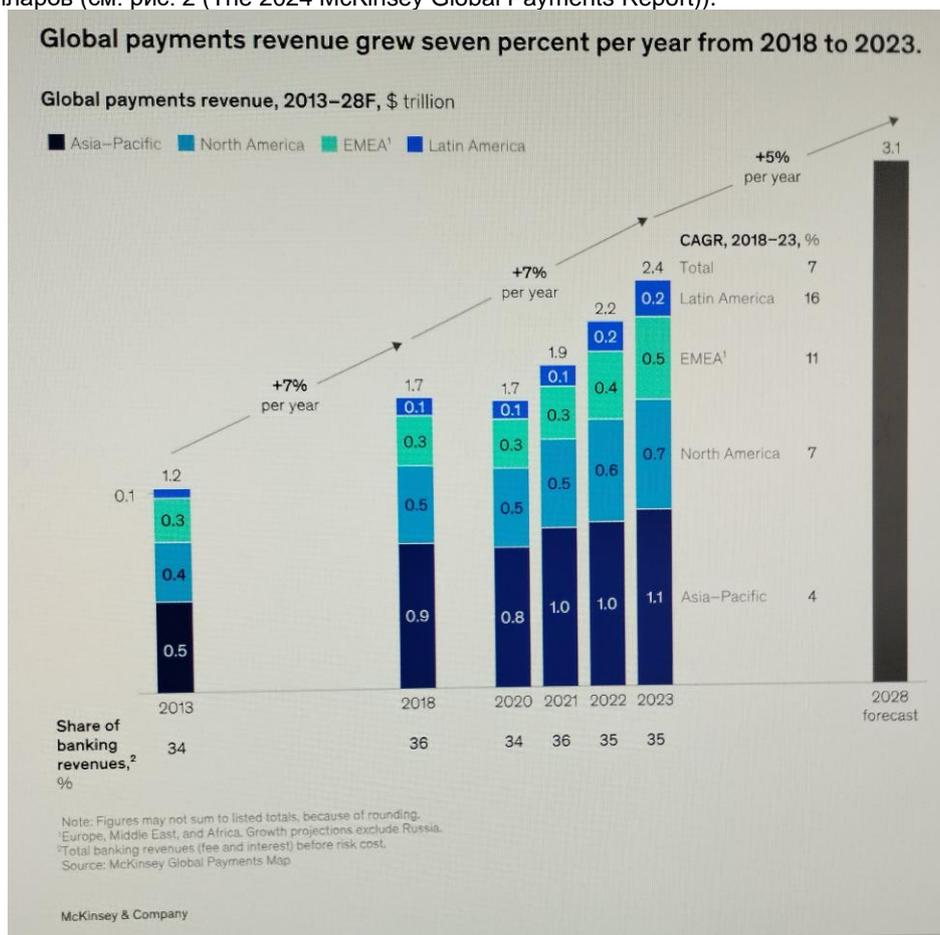


Рис.2. Динамика мировой индустрии электронных платежей.

В качестве еще одного примера можно сослаться на введение нового налога на товары и услуги в Индии с использованием электронной регистрации в платежной системе, что увеличило число налогоплательщиков на 50% менее чем за один год (Gupta, Keen, Shah, Verdier, 2017).

Приведенный выше анализ некоторых источников свидетельствует об активной дискуссии в сфере оптимизации налоговой системы России. Практика показывает, что к однозначному мнению исследователям пока не удается прийти. Однако практически все они сходятся во мнении, что развитие цифровизации налоговой системы и деятельности ее экономических субъектов будет способствовать повышению эффективности сбора налогов. Вместе с тем в ходе обсуждения открываются новые достоинства и недостатки как отдельных налогов, так и всей налоговой системы страны.

Материалы и методы

В качестве методологической базы использовалась совокупность общенаучных методов, а именно – логического, экономического анализа, статистического. При помощи логического метода были обоснованы причинно-следственные связи и факторы, оказывающие влияние на динамику уплаты налогов. Методы экономического анализа применялись для выявления динамики и структуры налоговых доходов федерального бюджета. На основе статистического метода сформирована релевантная информационная база, позволившая достичь поставленной цели исследования.

Авторы статьи полагают, что эффективность структуры налоговой системы принципиально зависит от структуры экономики конкретной страны. Для проверки этой гипотезы в настоящей работе выполнено сравнение работы НДС и налога с оборота при помощи цифрового моделирования динамики развития существенно различных по структуре экономик. Использование при цифровом моделировании компаративного метода позволяет не учитывать влияния множества деталей, которые не представляются существенными для нашего исследования.

Авторы придерживаются методологии планирования эксперимента, в соответствии с которой рассмотрена компаративная модель в максимально удаленных точках мыслимого диапазона отклонения параметров. Тогда с учётом главного отличия России как сверхобеспеченной ресурсами страны от стран Запада, который навязал нам НДС как якобы оптимальный налог, будем полагать следующее: 1-ая компаративная модель для страны А основывается на предположении, что она располагает готовыми ресурсами, на 1-2 порядка превышающими её текущую капитализацию, добывает сырьё и занимается его первым переделом, а 2-ая компаративная модель для страны В основывается на предположении, что она совсем не обладает ресурсами, но имеет технологическое превосходство и занимается всеми более высокими переделами.

Таможенные границы в системе из 2-х стран отсутствуют, следовательно, страна В имеет необходимый для выживания доступ к ресурсам страны А, через свободные обменные операции (неравноценные в силу технологического превосходства).

Тогда эволюционная вероятностная модель (Грачёв И.Д. и др., 2020) в предельно упрощенной форме может быть представлена в следующем виде:

$$A(i+1) = A(i) - \text{diag}(\xi_a) \cdot A(i) + \langle \xi_a(i) \rangle \cdot A(i) - \gamma_a \cdot A(i) \quad (1)$$

$$B(i+1) = B(i) - \text{diag}(\xi_b) \cdot B(i) + \langle \xi_b(i) \rangle \cdot B(i) - \gamma_b \cdot B(i) \quad (2)$$

$$\langle \xi_a(i) \rangle = (A(i) \cdot \xi_a^T + B(i) \cdot \xi_b^T + \mu \cdot \Pi) : (A(i) \cdot I^T + B(i) \cdot I^T + \Pi) \quad (3)$$

$$\langle \xi_b(i) \rangle = (A(i) \cdot \xi_a^T + B(i) \cdot \xi_b^T) : (A(i) \cdot I^T + B(i) \cdot I^T) \quad (4)$$

где $A(i)$, $B(i)$ – капиталы агентов стран А и В на i -ом цикле;

Π – монетизированные природные ресурсы;

μ – параметр, характеризующий доступность ресурсов.

Для определённости ξ_a и ξ_b можно рассматривать как вектора относительных себестоимостей генерирования «товара» агентами страны А и страны В. В этой интерпретации эволюционных уравнений (1) – (4) $\langle \xi_a(i) \rangle$ может рассматриваться как относительная рыночная стоимость товара, учитывающая доступность природных ресурсов в стране А, а $\langle \xi_b(i) \rangle$ – как рыночная стоимость товара в стране В, ресурсы которой либо ограничены, либо недоступны.

Результаты и обсуждение

Во всех цифровых экспериментах наилучшие значения векторов ξ_a и ξ_b совпадали, но размах ξ_a вдвое превышал размах ξ_b , с целью моделирования технологического превосходства страны В.

Остальные параметры в формулах (1) – (4) были подобраны так, чтобы страна А в отсутствие страны В при оборотном налоге 0,06 росла на старте на 3% в год.

Характерные результаты компаративного цифрового моделирования представлены рисунках 3 и 4.

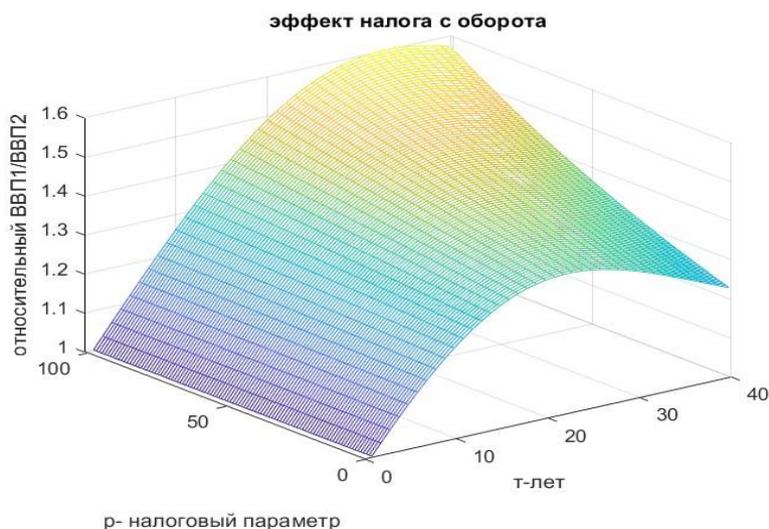


Рис. 3. Отношение ВВП сырьевой и высокотехнологичной стран (налог с оборота 0,06).

На рисунке 3 обе страны используют одинаковый оборотный налог в диапазоне $(0,03÷0,06)$. По оси Z рисунка представлены значения отношений $ВВП1(A)/ВВП2(B)$ за каждый год, при разном уровне налогообложения. Во всех модельных экспериментах при любом реалистичном уровне налогообложения страна (A) имеет явное превосходство по экономическому росту перед страной (B).

На рисунке 4 обе страны используют одинаковый НДС, выбранный исходя из сохранения бюджетных поступлений для страны B по сравнению с оборотным налогом. Однако для страны (B) с учетом предположения о высоких переделах распространяется не на всю цену товара.

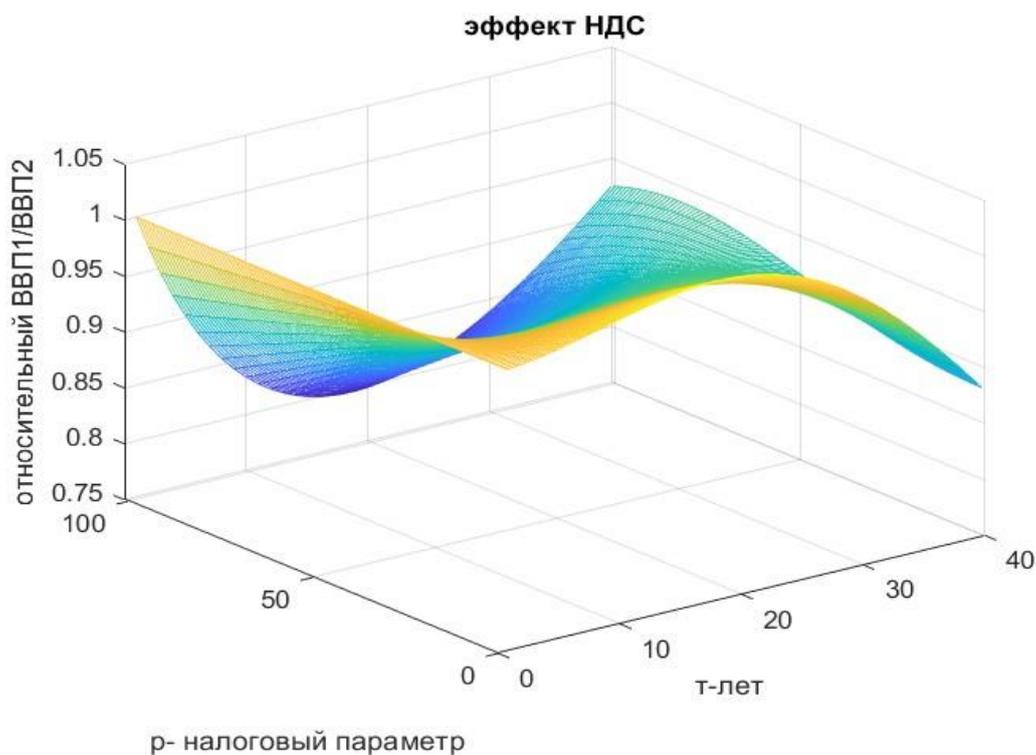


Рис. 4. Отношение ВВП сырьевой и высокотехнологичной стран (налог на добавленную стоимость 0,12).

В конкретном примере эффективная цена товара, на которую работает НДС, равна $\frac{1}{2}$ от полной.

Ещё раз отметим, что для чистоты моделирования мы предполагали, что собранные налоги не возвращаются в производство, а при формировании излишков сбрасываются в «стабилизационные фонды» по «бюджетному правилу».

График, приведенный на рисунке 4, переворачивает картину «с ног на голову». При замене налога с оборота равноценным для второй страны НДС, она имеет явное превосходство в экономическом росте по сравнению с хорошо обеспеченной ресурсами страной. Хотя надо отметить существенно более сложную картину зависимости компаративного отношения от уровня налогового пресса, что может быть предметом особого исследования, но не имеет отношения к нашей задаче.

Для наглядности результатов моделирования на рисунке 5 представлен экономический рост интересующей нас страны (А) при работе с оборотным налогом, а на рисунке 6 – при работе с эквивалентным налогом на добавленную стоимость.

То, что после 20-30 лет экономический рост страны А на рис. 5 прекращается, не должно смущать, так как мы преднамеренно для чистоты моделирования не вводили в модель ни технологические инновации, ни инновации, расширяющие ресурсную базу, а, следовательно, неявно закладывали «проедание» ресурсов на полуцикле Кондратьева. Тем не менее различие в развитии страны (А) при замене оборотного налога на НДС очевидно, и очевидно не в пользу НДС.

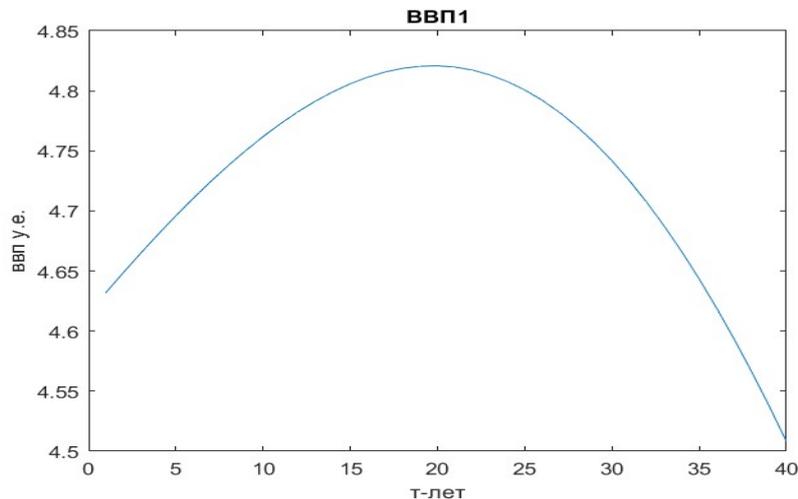


Рис. 5. Эволюция сырьевой страны (налог с оборота 0,06).

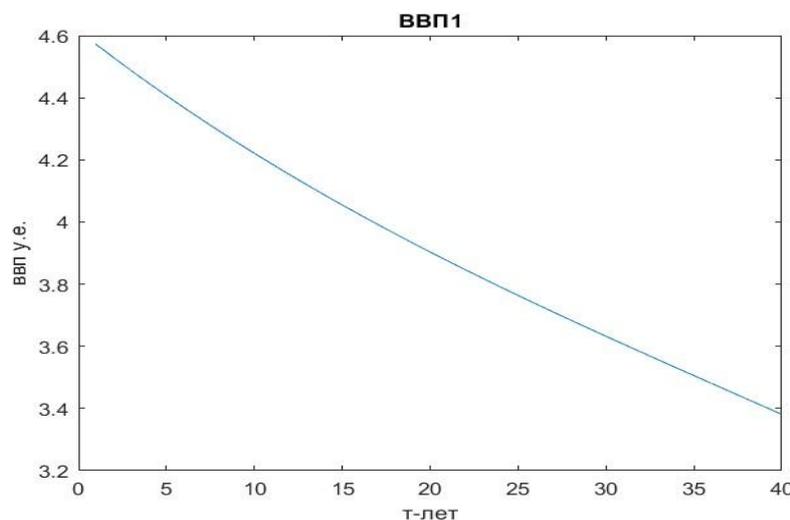


Рис. 6. Эволюция сырьевой страны (налог на добавленную стоимость 0,12).

В рамках цифрового моделирования замена оборотного налога на НДС для хорошо обеспеченной ресурсами страны (А), конкурирующей с ограниченной в ресурсах, но технологически опережающей ее страной (В), оказывает отрицательное влияние на эволюцию страны (А).

Разумеется, как и в большинстве случаев, цифровое моделирование нельзя считать строгим доказательством несостоятельности НДС для налоговой системы России как сверхобеспеченной ресурсами страны. Однако это дополнительный аргумент, который, в сочетании с тем, что другая хорошо обеспеченная ресурсами страна – США – вполне успешно обходится без НДС и не собирается его вводить, вполне убедителен.

Подтверждением этого можно считать тот факт, что по итогам 2024 года на первое место по объемам продаж вышла товарная группа «Летательные аппараты, космические аппараты, и их части» - объемы экспорта составят 129,8 млрд долл. США. Нефть расположилась на втором месте с 129,7 млрд долл. США, в пятерку также вошли нефтепродукты (121 млрд долл. США), легковые автомобили (60,7 млрд долл. США) и газ (60 млрд долл. США). Всего по итогам 2024 года США поставили за рубеж товаров на 2,08 трлн долл. США, из них на гражданские самолеты и нефть пришлось по 6,2%, на нефтепродукты - 5,8%, на легковые автомобили и газ - по 2,9% млрд долл. США (Названы главные американские, 2025).

Заключение

В связи с несоответствием экономических прогнозов МВФ и нашего финансового блока реальному росту России, а также активизацией научной дискуссии и практических изменений в США авторами была поставлена и решена задача цифровой экспериментальной проверки базовых утверждений, заложенных на основе Вашингтонского консенсуса в налоговую систему России.

Выполнено цифровое модельное сравнение работы используемого в России НДС и перспективной ситуации внедрения налога с оборота для предельно гиперболизированной ситуации бестамозженной системы двух стран. Одна из них обеспечена ресурсами с большим запасом, но ее экономика сосредоточена в основном на низких переделах. Вторая страна не имеет ресурсов, но обладает технологическим превосходством и высокими переделами.

С использованием компаративного подхода на цифровых моделях показано, что ресурсной стране выгоднее использовать налог с оборота, а стране, не обладающей ресурсами, оптимально использовать НДС самой и навязать его использование другой стране. Это позволяет утверждать, что выбор странами Европы использования НДС обоснован, отказ США от использования НДС обоснован, а выбор Россией использования НДС необоснован и экономически целесообразна его замена на категоризованный налог с оборота.

Список литературы

1. Бачурин Д.Г. Институциональные парадоксы НДС в условиях развивающейся экономики переходного периода // Актуальные проблемы экономики и права. 2016. №2 (38) [Электронный ресурс]. URL – [https:// cyberleninka.ru/article/n/institutsionalnye-paradoksy-nds-v-usloviyah-razvivayuyshchey-ekonomiki-perehodnogo-perioda](https://cyberleninka.ru/article/n/institutsionalnye-paradoksy-nds-v-usloviyah-razvivayuyshchey-ekonomiki-perehodnogo-perioda) (Дата обращения: 20.02.2025).
2. Буланова В.В. Отличия налога на добавленную стоимость от налога с продаж // Молодой ученый, 2021. № 13(355). С. 141-143. [Электронный ресурс]. – URL: <https://moluch.ru/archive/355/79561/> (дата обращения: 19.02. 2025).
3. Бухарова С.А., Калмышова К.Д. Сравнительный анализ достоинств и недостатков двух основных моделей потребления: НДС и налога с продаж // Приднепровский научный вестник, 2017. № 4. С. 6-8.
4. Грачёв И.Д., Грачёв Д.И., Ларин С.Н., Ноакк Н.В. Оценка экономических результатов различных вариантов карантинных с использованием комбинированной цифровой экономико-эпидемиологической модели // Экономика и предпринимательство, 2020. Вып.14. №2(115). С.902-909. <https://doi.org/10.34925/EIP.2020.115.2.182>
5. Исполнение федерального бюджета и бюджетов бюджетной системы Российской Федерации за 2023 год [Электронный ресурс]. – URL – https://minfin.gov.ru/common/upload/library/2024/06/main/Kniga_2023.pdf?ysclid=m795qi59uy183543769 (Дата обращения: 20.02.2025).
6. Киреев А.П., Содномова С.К. Трансформация налога на добавленную стоимость и современные практики улучшения его собираемости // Вестник Тюменского государственного университета. Социально-экономические и правовые исследования, 2020. Том 6. № 3(23). С. 273-294. DOI: 10.21684/2411-7897-2020-6-3-273-294.
7. Магомадов Ш.А., Джанаралиев Ш.Ш. Налог на добавленную стоимость: особенности и отличия от налога с продаж // Экономика и бизнес: теория и практика, 2019. № 4. Т. 3. С. 74-76.
8. Минаева Е.В., Аношина Ю.Ф. Налогообложение прибыли организаций как регулятор макроэкономических процессов. // Государственное и муниципальное управление. Ученые записки. 2019. № 3. С. 122-126.
9. Названы главные американские экспортные товары [Электронный ресурс]. URL – <https://ria.ru/20241013/bayden-1977801217.html?ysclid=m7934le2k6880243252> (Дата обращения: 17.02.2025).
10. НДС в странах мира в 2024 году. VisaSam [Электронный ресурс]. URL – <https://visasam.ru/emigration/vybor/nds-v-mire.html>. (Дата обращения: 19.02. 2025).
11. Пансков В.Г. Необходим новый подход к налоговому регулированию российской экономики // ЭТАП: Экономическая Теория, Анализ, Практика. 2020. № 1. С. 26-46.
12. Скобликов Е.А. О необходимости проведения кардинальной реформы налоговой системы России // Модели, системы, сети в экономике, технике, природе и обществе. 2014. №4(12). [Электронный ресурс]. URL – [https://cyber leninka.ru/article/n/o-neobhodimosti-provedeniya-kardinalnoy-reformy-nalogovoy -sistemy-rossii](https://cyberleninka.ru/article/n/o-neobhodimosti-provedeniya-kardinalnoy-reformy-nalogovoy-sistemy-rossii) (Дата обращения: 20.02.2025).

13. Чехов А.В. НДС и «налог с оборота». Сравнительная характеристика. // RUSSIAN JOURNAL OF MANAGEMENT, 2021. Том 9. № 1. С.76-80. DOI: 10.29039/2409-6024-2021-9-1-76-80.
14. Шамаев С.А. Налог с оборота, налог с продаж и налог на добавленную стоимость как различные подходы к применению универсального акциза // Налоги и налогообложение. 2024. № 2. DOI: 10.7256/2454-065X.2024.2.70388 EDN: HBYPAR URL: https://nbpublish.com/library_read_article.php?id=70388.
15. Gupta S., Keen M., Shah A., Verdier G. Digital Revolutions in Public Finance / International Monetary Fund, 2017. – 362 p.
16. The 2024 McKinsey Global Payments Report. Global payments in 2024: Simpler interfaces, complex reality. [Электронный ресурс]. URL – <https://www.mckinsey.com> (Дата обращения: 20.02.2025)..

*Грачев Иван Дмитриевич – д.э.н., к.ф.-м.н.,
главный научный сотрудник ЦЭМИ РАН г. Москва
ORCID 0000-0003-1815-5898
ldg@mail.ru*

*Ларин Сергей Николаевич, к.техн.н.,
ведущий научный сотрудник ЦЭМИ РАН г. Москва
ORCID 0000-0001-5296-5865
sergey77707@rambler.ru
г. Москва*

*Ноакк Наталия Вадимовна – к.психол.н.,
ведущий научный сотрудник ЦЭМИ РАН
ORCID 0000-0001-8696-5767,
n.noack@mail.ru*

Ключевые слова: налог с оборота и НДС, ресурсная и высокотехнологичная страны, компаративное цифровое моделирование экономик.

Ivan Grachev, Sergey Larin, Natalia Noakk, Digital modeling of VAT and turnover tax

Keywords: turnover tax and VAT, resource and high-tech countries, comparative digital modeling of economies.

DOI: 10.34706/DE-2025-01-09
JEL classification C52, C65, E62, H21

Abstract: The changes taking place in the global economic system have called into question all the axioms of the Washington Consensus type and updated scientific research, in particular, on tax and customs systems. These problems are also important for our country, but, unfortunately, approaches to their solution are mostly verbal and far from universal. In this paper, comparative digital modeling of the comparative effectiveness of turnover tax and value-added tax for diametrically different resource-rich countries is performed. One of them is provided with resources with a good reserve for the future, but its economy is focused on low redistribution. The second one has no natural resources at all, but it has technological superiority, which is expressed in the dominance of high redistribution. Digital models show that with all realistic levels of technological superiority and taxation levels, as well as the complete absence of customs duties, the choice of the optimal tax for a country with high resource availability and a high-tech country is diametrically different. The results obtained in the course of digital modeling additionally, along with traditions and history, substantiate the validity of the hypothesis about the justified refusal of the USA from VAT. Today, the United States, like Russia, belongs to resource-rich countries, since three of the top four positions in its exports are occupied by oil, refined products and gas. Based on the results of digital modeling, it is concluded that Russia currently has even more reasons to refuse VAT than the United States.

2. ОБЗОРЫ

2.1. Советские истоки компьютерной эры в Китае

Владимир А. Китов и Ольга В. Китова,
РЭУ им. Г.В. Плеханова. Москва, Россия

Статья посвящена истории помощи советских ученых, прежде всего Анатолия Ивановича Китова, Исаака Семёновича Брука, Сергея Алексеевича Лебедева и Николая Андреевича Криницкого, китайским коллегам в становлении компьютерного производства. В статье прослеживается связь первых достижений китайских ученых в области информатики с историей создания вычислительной техники в СССР.

Её актуальность обусловлена тем, что недавние исследования в области истории вычислительной техники, опубликованные на английском языке западными историками, могут создать у ученых впечатление, что влияние советских ученых на китайских коллег на начальном этапе компьютерной эры в Китае было весьма незначительным.

Исследование основано на анализе научных работ по истории вычислительной техники, материалах Виртуального компьютерного музея, фондах Государственного политехнического музея России, архивах Российского экономического университета им. Г.В. Плеханова, а также на личных встречах авторов с китайскими учеными и переводчиками.

Преамбула

Несколько дней назад информационное пространство захватило громкое заявление руководителя китайского стартапа DeepSeek Лян Вэньфэна о том, что их ИИ был создан на основе советских технологий, в частности системы ОГАС (Общегосударственная автоматизированная система учёта и обработки информации). В интервью на YouTube-канале американского блогера Лекса Фридмана он отметил, что без этой системы DeepSeek не смог бы достичь успеха и догнать западные аналоги. Предприниматель отметил, что он никогда не забывал о вкладе советских учёных в научно-технический прогресс и важности их наследия для современного Китая, а также объявил о намерении открыть в Пекине музей истории, посвящённый научным достижениям СССР в области вычислительной техники. Новость тиражировалась, в частности, в эфире программы «Вести» на «России 1» и ряде Telegram-каналов популярных политиков. Впоследствии появились публикации о том, что эта новость является шуткой сатирического издания ИА «Панорама». Не имея намерения втягиваться в расследование инцидента, авторы этой статьи все же решили покопаться в забытых архивах. Насколько к данной истории применима пословица «нет дыма без огня», судить читателю.

Первые шаги цифровизации в Китае

Сегодня в Китае стремительно развивается информационное общество и цифровая экономика. Страна постепенно выходит в лидеры инноваций, неразрывно связанных с цифровыми технологиями. В этой связи интересно вспомнить, что первые шаги Китая в области компьютеризации были бы невозможны без помощи коллег из СССР.

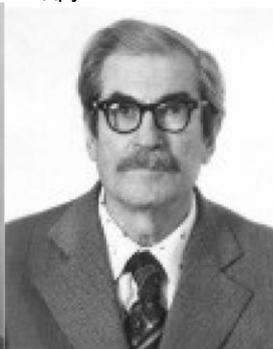
В 1940-е/1950-е годы мировыми странами-пионерами в области компьютеров были Соединённые Штаты Америки, Великобритания и Советский Союз. Также общеизвестно, что выдающуюся роль в создании первых в мире компьютеров сыграл немецкий учёный Конрад Цузе. Серьёзный интерес в Китае к созданию своих собственных компьютеров возник во второй половине 1950-х годов. Но из-за идеологических разногласий научно-техническое взаимодействие Китая с капиталистическими странами в тот период времени было практически сведено к нулю. Объективно, первые шаги в области компьютеризации китайские ученые могли сделать только при поддержке коллег из дружественного СССР.



И.С. Брук



С.А. Лебедев



Н.А. Криницкий



А.И. Китов

В рамках Китайской академии наук в 1950-е годы был создан Институт вычислительной техники. Китайские учёные, работающие во всех базовых направлениях науки, интенсивно изучали русский язык.

При непосредственном участии советских специалистов были запущены в производственную эксплуатацию два первых китайских компьютера. Созданная в августе 1958 года, китайская ЭВМ «103» была полным аналогом советской ЭВМ «М-3», созданной в СССР коллективом специалистов под руководством советского компьютерного пионера Семёна Исааковича Брука. Следующая китайская ЭВМ «104» была запущена в октябре 1959 года. Она являлась полным аналогом ЭВМ «БЭСМ-2», разработанной в СССР в Институте точной механики и вычислительной техники под руководством другого советского компьютерного пионера Сергея Алексеевича Лебедева.

Параллельно с запуском первых китайских компьютеров, из СССР в Китай поступило подавляющее большинство литературы по компьютерам и программированию. В то время из двенадцати переведенных монографий и брошюр по вычислительной технике одиннадцать были переведены с русского на китайский язык. Первая из этих монографий, написанная А.И. Китовым, была издана в СССР в первой половине 1956 года. Это была трехсот-страничная книга «Электронные цифровые машины» [5], которая была издана в Китае уже в октябре 1958 года. И буквально через месяц, в ноябре 1958 года, второе издание этой книги появилось в магазинах Китая. Всего было издано около шести тысяч экземпляров этой пионерской монографии Китова в Китае.

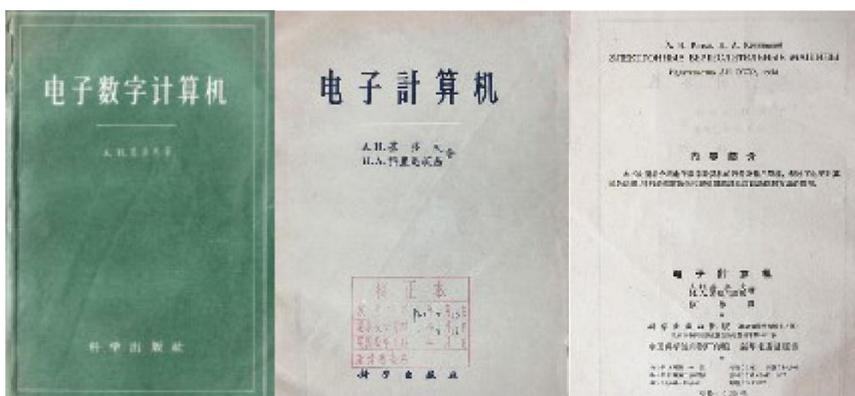
Переводчиком монографии «Электронные цифровые машины» был Чжан Вэй. Автор этой статьи Владимир Китов в 2012 году находился в Китае для чтения лекций в университете Цинь Хуа и встретился с Чжан Вэем в Пекине. Чжан Вэй рассказал ему: «Перевод монографии А.И. Китова „Электронные цифровые машины“ моя жена Юй Гуйчжи и я начали в 1957 году. Мы испытали огромные трудности, прежде всего, в области понимания принципов работы компьютера и терминологии, так как, как и в самом СССР, так и в Китае эта монография была первой. В Китае до этих пор не было других книг в области программирования, компьютеров и их приложений».

Вторая монография А.И. Китова была опубликована в Китае в 1961 году. Она называлась «Электронные вычислительные машины» и была написана им совместно с выдающимся советским ученым в области программирования Николаем Андреевичем Криницким. В СССР первое издание этой монографии было выпущено в 1958 году издательством Академии наук СССР под редакцией замечательного ученого Анатолия Алексеевича Дородницына, о котором следует особо упомянуть. В 1955 году А.А. Дородницын стал основателем и первым директором Вычислительного центра Академии наук СССР (до 1989 года). Сейчас этот Вычислительный центр носит его имя. С 1989 по 1994 годы А.А. Дородницын был почетным директором и научным руководителем этого центра. В 1960 году А.А. Дородницын стал одним из инициаторов создания IFIP, а с 1968 по 1971 год занимал пост председателя IFIP. Перевод монографии «Электронные вычислительные машины» на китайский был опубликован в Китае в марте 1961 года. В декабре того же года, то есть всего через восемь месяцев после первого издания на китайском, в Китае было выпущено второе издание этой книги. Общий тираж двух изданий монографии «Электронные вычислительные машины» составил 18 000 экземпляров.

Общий тираж этих двух монографий А.И. Китова — «Электронные цифровые машины» и «Электронные вычислительные машины» — в Китае значительно превысил тираж всех остальных книг по вычислительной технике, изданных в Китае в рассматриваемый период. По мнению профессора Джан Вэя и его китайских коллег, причины феноменального успеха этих двух монографий А.И. Китова в Китае заключаются в том, что их содержание было чрезвычайно насыщенным, очень ясным и четко изложенным для понимания первого поколения китайских специалистов и студентов, осваивающих информационные технологии.

Следует упомянуть, что профессор Мичиганского университета (США) Джон Карр (John Carr) в своей монографии «Лекции, прочитанные в Мичиганском университете», опубликованной в США летом 1958 года, о книге А.И. Китова «Электронные цифровые машины» пишет: «Очевидно, наиболее полное изложе-

ние вопросов программирования для электронных цифровых вычислительных машин, содержащее подробные примеры и анализ как ручного, так и автоматического программирования, представлено в книге А.И. Китова. Некоторые разделы этой книги были переведены на английский и могут быть получены через Ассоциацию вычислительной техники» [7]. В СССР эта монография профессора Джона Карра была опубликована в 1963 году [6].



Книги А.И. Китова «Электронные цифровые машины» (1956) и А.И. Китова и Н.А. Криницкого «Электронные вычислительные машины» (1958), опубликованные в Китае соответственно в 1958 г. и в 1961 г.

Несомненный исторический интерес представляет статья китайского историка, профессора Оу Бау [1], в которой раскрываются ряд важных моментов истории китайской компьютеризации в первый период её становления с 1953-го года по 1966-й год. Статья стала итогом кропотливой работы профессора Оу Бау в Государственной библиотеке Китая, в библиотеке Пекинского университета Цинь Хуа, а также с личным архивом супругов Юй Гуйчжи и Чжан Вэй. Профессор Чжан Вэй дал два интервью профессору Оу Бау 30 марта и 6 апреля 2012 года. Как он вспоминает: «Если бы Юй Гуйчжи и я не были бы мужем и женой, мы не смогли бы так быстро перевести эту замечательную книгу А.И. Китова «Электронные цифровые машины». После её публикации эта книга сразу же начала использоваться всеми китайскими университетами в качестве учебника. В 1960 году я продолжил обучение в Китайском университете науки и технологии, где наш преподаватель постоянно использовал эту книгу А.И. Китова при чтении своих лекций. Мне приятно вспоминать, как наш преподаватель и студенты радостно хлопали меня по спине, когда узнали, что именно я перевел книгу Китова с русского на китайский». Трудолюбивые переводчики Юй Гуйчжи и Чжан Вэй учились вместе в Институте русского языка в Харбине. К сожалению, во время Культурной революции в Китае талантливые специалисты и переводчики Чжан Вэй и Юй Гуйчжи были привлечены к суду и сосланы в отдалённую сельскую местность Китая на десять лет для «перевоспитания через труд».

За весь первоначальный период формирования китайской компьютерной индустрии, то есть до начала Культурной революции, в Китае было издано двадцать семь переводов иностранных книг и брошюр по вычислительной технике и программированию. Общий объем этих переводов составил 4198 страниц. Среди этих двадцати семи иностранных книг и брошюр восемнадцать публикаций были из Советского Союза. Из оставшихся девяти публикаций, переведенных на китайский язык, стоит отметить, что восемь публикаций были переведены с английского языка и одна — с японского.

На начальном этапе компьютерной эры в Китае наибольшее внимание привлекали публикации, посвященные основным концепциям вычислительной техники и программирования, таким как структура, принципы создания компьютеров и основные области их применения. Очевидно также, что поскольку первые два китайских компьютера «103» (1958 год) и «104» (1959 год) были точными копиями советских компьютеров «М-3» (1956) и «БЭСМ-2» (1957) соответственно, описания устройств этих двух советских компьютеров пользовались большим спросом среди китайских специалистов по вычислительной технике. Поэтому неудивительно, что в 1959 году в Китае был опубликован китайский перевод коллективной книги «Малогобаритный электронный компьютер М-3» В.В. Беленского и его соавторов. В 1963 году в Китае был опубликован перевод книги «Цифровая техника и вычислительные устройства» создателя компьютера М-3 Исаака Брука, изданной в СССР в 1959 году. В 1962 году в Китае была опубликована книга «Компьютерная техника и её применение», отредактированная создателем БЭСМ-2 С.А. Лебедевым. Эта книга была издана в СССР в 1959 году. Не обошлось и без популярной литературы. В 1958 году в Китае была издана простая публичная брошюра «Думающие машины» М.Г. Рейнберга, ранее изданная в СССР в издательстве «Детгиз». Из специальной литературы в 1960 году в Китае была опубликована книга В.Д. Моисеева «Автоматические компьютеры и их применение на железнодорожном транспорте», изданная в СССР в 1957 году. В стартовый период китайской компьютерной эры, т.е. до 1960-го года, переводы советской компьютерной литературы на китайский язык имели подавляющий характер – одиннадцать публикаций из двенадцати. В этот период основной объем первых советских компьютерных книг (шесть книг, 1133 страниц) был переведен с русского языка на китайский замечательными специалистами и переводчиками супругами Юй Гуйчжи и Чжан Вэем.

В 1960 году по политическим причинам произошел принципиальный разрыв в отношениях между Советским Союзом и Китаем, что повлекло за собой прекращение взаимодействия почти во всех сферах научно-технического сотрудничества. Это притормозило развитие китайской компьютерной индустрии. Однако самые важные сектора национальной экономики Китая, такие как геология, экономика, нефтедобыча, прогнозирование погоды и военная сфера (прежде всего создание баллистических ракет и ядерного оружия) требовали разработки новых компьютерных моделей. Поэтому в последующие несколько лет на прочном фундаменте из предыдущего плодотворного опыта сотрудничества с советскими специалистами в Китае были созданы компьютеры серии "107" в 1960 году, "119" в 1964 году, "1092" в 1965 году и "1093" в 1967 году.

С 1966 года Китай пережил десятилетний трудный период, который получил название «Культурная революция». В это время в стране практически не разрабатывались новые компьютеры, поэтому можно утверждать, что базовый период формирования и развития китайской компьютерной индустрии, приходящийся на период с 1953 по 1967 год, проходил под эгидой российской школы Computer Science.



Опубликованные в Китае книги
И.С. Брука и С.А. Лебедева

Заключение

Два первых китайских компьютера («103» (1958 год) и «104» (1959 год)) были точными копиями советских компьютеров «М-3» (1956) и «БЭСМ-2» (1957) соответственно. На базовом этапе формирования отрасли компьютерная литература и техническая документация из Советского Союза были основным источником компьютерных знаний для китайских специалистов, студентов и ученых. Инженерная помощь специалистов из СССР и советская компьютерная литература оказали огромную помощь китайским коллегам, что стало решающим положительным фактором в формировании китайской компьютерной индустрии в целом.

Список литературы:

1. Оу Бао. Первые публикации о компьютерах и программировании в Китае на заре компьютерной эры с 1953 по 1967 годы // Институт истории естествознания и техники им. С.И. Вавилова РАН. Ежегодная научная конференция. Т. 2. С. 809-813. Москва, 2012.
2. Оу Бао, Китов В.А., Шилов В.В. Первые советские книги о компьютерах в Китае // Институт истории естествознания и техники им. С.И. Вавилова РАН. Ежегодная научная конференция. Т. 2. С. 349-351. Москва, 2013.
3. Китов В.А., Прохоров С.П. Первые отечественные публикации по кибернетике, программированию, вычислительным машинам и их приложениям // Институт истории естествознания и техники им. С.И. Вавилова РАН. Ежегодная научная конференция. Т. 2. С. 778-781. Москва, 2012.
4. Шилов В.В. Страницы жизни и научной деятельности Анатолия Ивановича Китова // Труды Свободного экономического общества России. Т. 143. Москва, 2010. (На русском языке).
5. Китов А.И. Электронные цифровые машины // Советское радио. М., 1956. // http://www.computer-museum.ru/books/kitov_ecm.htm.
6. Карр Дж. Лекции по программированию. Перевод, под редакцией В.М. Курочкина. Издательство иностранной литературы. Москва, 1963.
7. Carr, John Lectures given at the university of Michigan, Summer, 1958.

Дополнительные источники

1. И.С. Брук: <https://www.computer-museum.ru/galglory/bruk.htm>
2. С.А. Лебедев: <https://www.computer-museum.ru/galglory/lebedev0.htm>
3. А.И. Китов. Википедия: https://en.wikipedia.org/wiki/Anatoly_Kitov .
4. Н.А. Криницкий: https://student.mirea.ru/80_years/walk_of_fame_mirea/?ELEMENT_ID=6052

References in Cyrillics

1. Ou Bao. Pervye publikacii o komp'yuterax i programmirovanii v Kitae na zare komp'yuter-noj èry` s 1953 po 1967 gody` // Institut istorii estestvoznaniya i texniki im. S.I. Vavilova RAN. Ezhegodnaya nauchnaya konferenciya. T. 2. S. 809-813. Moskva, 2012.
2. Ou Bao, Kitov V.A., Shilov V.V. Pervye sovetskie knigi o komp'yuterax v Kitae // Institut istorii estestvoznaniya i texniki im. S.I. Vavilova RAN. Ezhegodnaya nauchnaya konferenciya. T. 2. S. 349-351. Moskva, 2013.
3. Kitov V.A., Proxorov S.P. Pervye otechestvenny`e publikacii po kibernetike, programmirovaniyu, vychislitel`ny`m mashinam i ix prilozheniyam // Institut istorii estestvoznaniya i texniki im. S.I. Vavilova RAN. Ezhegodnaya nauchnaya konferenciya. T. 2. S. 778-781. Moskva, 2012.
4. Shilov V.V. Stranicy zhizni i nauchnoj deyatel`nosti Anatoliya Ivanovicha Kitova // Trudy` Svobodnogo èkonomicheskogo obshhestva Rossii. T. 143. Moskva, 2010. (Na russkom yazy`ke).
5. Kitov A.I. Èlektronny`e cifrovyye mashiny` // Sovetskoe radio. M., 1956. // http://www.computer-museum.ru/books/kitov_ecm.htm.
6. Karr Dzh. Lekcii po programmirovaniyu. Pervod, pod redakciej V.M. Kurochkina. Izdatel`stvo inostrannoj literatury`. Mosk6va, 1963.

Additional sources

1. I.S. Bruk: <https://www.computer-museum.ru/galglory/bruk.htm>
2. S.A. Lebedev: <https://www.computer-museum.ru/galglory/lebedev0.htm>
3. A.I. Kitov. Википедия: https://en.wikipedia.org/wiki/Anatoly_Kitov .
4. N.A. Krinitskiy: https://student.mirea.ru/80_years/walk_of_fame_mirea/?ELEMENT_ID=6052

Благодарности

Данное исследование выполнено в рамках государственного задания в сфере научной деятельности Министерства науки и высшего образования РФ на тему "Модели, методы и алгоритмы искусственного интеллекта в задачах экономики для анализа и стилизации многомерных данных, прогнозирования временных рядов и проектирования рекомендательных систем", номер проекта FSSW-2023-0004.

This research was performed in the framework of the state task in the field of scientific activity of the Ministry of Science and Higher Education of the Russian Federation, project "Models, methods, and algorithms of artificial intelligence in the problems of economics for the analysis and style transfer of multidimensional datasets, time series forecasting, and recommendation systems design", grant no. FSSW-2023-0004.

*Владимир Анатольевич Китов,
к.т.н., доцент РЭУ имени Г.В. Плеханова
(115054, Россия, Москва, Стремянный переулок 36, корпус 9)
kitov.va@rea.ru*

*Ольга Викторовна Китова,
доктор экономических наук, доцент, зав. кафедрой информатики РЭУ им. Г.В. Плеханова (115054,
Россия, Москва, Стремянный переулок 36, корпус 9),
ORCID: 0000-0002-1820-0954,
kitova.ov@rea.ru*

Ключевые слова

Электронные цифровые машины; информационное общество, цифровая экономика.

Vladimir A. Kitov, Olga V. Kitova, The Soviet Origins of the Computer Age in China

Keywords

Electronic digital machines; information society, digital economy.

DOI: 10.34706/DE-2025-01-10

JEL classification: C63 «Вычислительная техника».

Abstract

The article is devoted to the history of the assistance of Soviet scientists, primarily Anatoly Ivanovich Kitov, Isaac Semenovich Bruk, Sergei Alekseevich Lebedev and Nikolai Andreevich Krinitsky, to Chinese colleagues in the development of computer production. The article traces the connection of the first achievements of Chinese scientists in the field of computer science with the history of the creation of computer technology in the USSR.

Its relevance is due to the fact that recent research in the field of the history of computing technology, published in English by Western historians, may give scientists the impression that the influence of Soviet scientists on their Chinese colleagues at the initial stage of the computer era in China was very insignificant.

The research is based on an analysis of scientific papers on the history of computing, materials from the Virtual Computer Museum, the collections of the State Polytechnic Museum of Russia, the archives of the Plekhanov Russian University of Economics, as well as personal meetings of the authors with Chinese scientists and translators.

3. МНЕНИЯ

УДК: 336.71

3.1. Банк-как-Сервис – параллельная реальность финансовых экосистем

Фролов В.Н., Романчук А.П., Дорофеев А.П.
АО «Цифровая Динамика», Екатеринбург, Россия

В статье рассмотрена новейшая технология Банк-как-Сервис (Banking-as-a-Service, принятое сокращение BaaS) в финансовой индустрии, история платёжных решений, предпосылки появления BaaS, особенности применения, преимущества использования этого решения для банков и небанковских компаний, а также вызовы для консервативного банковского сообщества.

Эпоха перемен

Сегодня во всём мире и в России, пожалуй, в первую очередь, сложилась уникальная возможность кардинального переустройства банковской сферы, перехода на принципиально новую модель финансовой системы.

В мире никогда ничего не бывает вечного. Всё течёт, всё изменяется. В России уже не будет приватизации по образу и подобию 90-х годов, и не станут в одночасье появляться миллиардеры, обогатившиеся на разделе советского приданого. Так и с традиционной банковской индустрией, которая столкнулась с новыми вызовами, основанными, прежде всего, на самых новейших технологиях.

Классические «аналоговые» банки были автоматизированы достаточно давно, и многие из них вынуждены работать в уже устаревшей сейчас IT-архитектуре. Внутри таких банков установлено большое количество разнообразного ПО, которое часто бывает разрозненно и плохо интегрировано как между собой, так и с внешними решениями: отсутствует бесшовность между системами, повсеместно нет открытого API (интерфейса прикладного программирования). Средства же разработки продуктов часто весьма архаичны. Например, в устаревшей банковской IT-архитектуре возникает дублирование продуктов и информации в разных системах: АБС (автоматизированной банковской системе), ДБО (дистанционном банковском обслуживании), карточном процессинге, которые требуется поддерживать и обновлять синхронно. Часто многие банки добавляют в свою архитектуру ESB-системы (Enterprise Service Bus, шина данных), что только усугубляет проблему дублирования и увеличивает расходы.

Традиционные банки работают в режиме 24x7, и модернизировать их IT-архитектуру без остановки платёжных процессов крайне тяжело, а скорее всего даже и невозможно. У банкиров есть только очень малое время для простоя операций, иначе недовольные сервисом клиенты начнут массово переходить в другие, более стабильные с технической точки зрения банки. Современные технологии безостановочных обновлений ПО банкам часто недоступны прежде всего из-за устаревшей архитектуры.

Есть ещё несколько факторов, которые лежат тяжким грузом на плечах ветеранов банковского дела. Это высокие накладные затраты по причине громоздкости и неэффективности процессов, необходимость поддерживать устоявшуюся бизнес-модель, в т.ч. операционные офисы, многочисленный персонал, постоянное давление со стороны регулятора плюс высочайшие требования к информационной безопасности.

Также, как мы убедились, многие классические банки в России оказались не готовы к таким вызовам как глобализация, тотальный переход в онлайн на фоне пандемии и всемирного бойкота со стороны западных производителей ПО.

В этой обстановке на передний план, в первую очередь, выходят так называемые *необанки*, банки без офисов, без бумажного документооборота. Не обременённые наследием старого ПО и бюрократией, они часто стартуют с более современных технологий и привлекают квалифицированных IT-специалистов. Процессы у таких необанков оказываются проще и гибче, чем у патриархов банковской индустрии: быстрее вносятся изменения, оперативно адаптируются сервисы для клиентов, бесшовно интегрируются сторонние финтех-решения и т. д. А также зачастую к необанкам на старте их финансового бизнеса предъявляются более простые требования регуляторов. Однако, очень часто необанки строят свои сервисы поверх какой-либо существующей банковской инфраструктуры, замещая лишь пользовательские интерфейсы на современные. Редко кто из необанков пытается создавать полноценную архитектуру цифрового банка.

На Западе появились такие необанки, как Simple, Moven, Hello Bank, Bank Direct, Atom Bank, Monzo, Revolute, Nubank и другие, которые активно вытесняют аналоговых старожилов банковской индустрии.

Для примера, 9 декабря 2021 года бразильский Nubank вышел на IPO на NYSE, оценка составила 45 млрд долларов¹, что сделало его самым дорогим небанком.

Continued Growth of One of the World's Largest Digital Banking Platforms

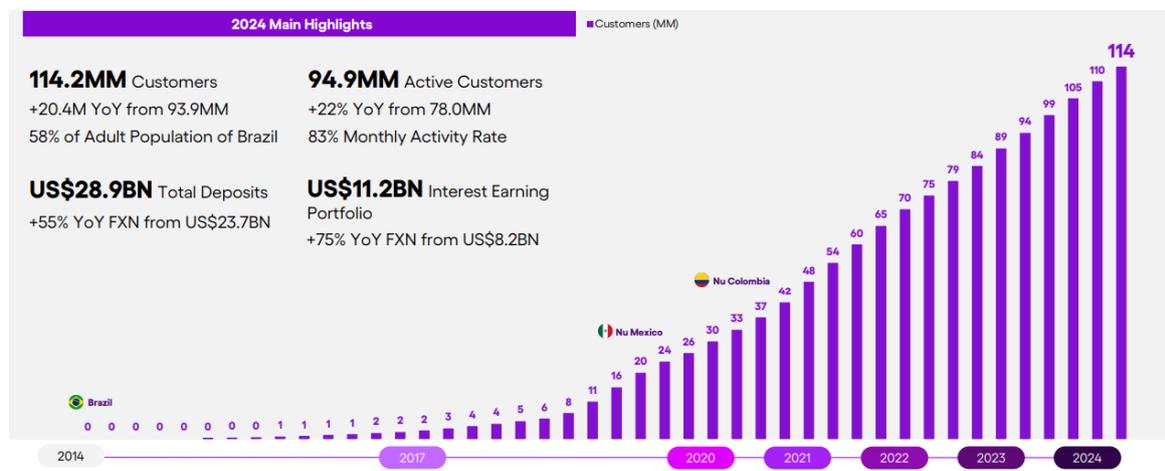


Рис 1. Результаты деятельности бразильского Nubank по итогам 2024г.
Источник: <https://www.investidores.nu/en/financials/results-center/>

А в российском банковском секторе вышли на рынок свои новички: основанный в 2006 г. Тинькофф Банк (в 2016 г. международная консалтинговая компания Frost & Sullivan признала его самым крупным небанком в мире², в 2024 году он переименован в «Т-Банк»), Модульбанк, Точка, Рокетбанк, Touch Bank, TalkBank. К 2016 году, по данным исследовательского центра Burnmark, в мире было уже около 70 небанков, в том числе 40 в Великобритании, 8 — в Индии, 5 — в США, 4 — во Франции, 3 — в Германии.

Вслед за чисто банковскими игроками на финансовый рынок стали выходить IT-компании, такие рыночные гиганты, как Apple с Apple Pay Cash, Alibaba с AliPay, TenCent с WeChat Pay, Google с Wallet, Xiaomi с Mi Pay и многие другие на Западе и Востоке. В России в качестве примера можно привести IT-корпорацию Яндекс, маркетплейсы Wildberries и Озон, у которого финтех-направление уже превышает более 30% в общей выручке бизнеса, а результаты завершившегося 2024 г. озадачили банковское сообщество своей успешностью. Выручка Озон-финтех увеличилась почти втрое до 93,3 млрд рублей, прибыль до налогов выросла в 2,3 раза и превысила 26,5 млрд. Число активных клиентов на конец декабря составило 30,3 млн человек, сумма их средств на вкладах и счетах увеличилась в 3,6 раза, до 193 млрд рублей³.

Телекомы тоже не дремлют. Скажем, корпорация МТС давно строит свою экосистему вместе с принадлежащим ей банком. Есть своё финтех-направление и у Билайна. Даже девелоперы и те занялись банковским бизнесом. Если бы не внезапно развернувшийся кризис на рынке жилья, то ГК «Самолёт» уже в 2025г. взлетела бы со своим финтех-проектом на базе собственного банка⁴.

Пожалуй, наиболее яркий пример для B2B сектора в России - это СКБ «Контур», который менее года назад вывел на финансовый рынок свой одноимённый банк, построенный на классической платформе Банк-как-сервис. Особенностью цифрового банка «Контур. Банк» является бесшовная интеграция со своими многочисленными цифровыми сервисами - электронным документооборотом Контур.Диадок, онлайн-бухгалтерией Контур.Эльба и пр.

¹ Bloomberg: Brazil's Nubank Jumps in Debut After IPO Raises \$2.6 Billion, URL: <https://www.bloomberg.com/news/articles/2021-12-09/brazil-s-nubank-jumps-in-debut-after-raising-2-6-billion-ipo>

² Т-Банк: Тинькофф Банк — самый большой независимый онлайн-банк в мире, URL: <https://www.tbank.ru/about/news/06102016-tinkoff-named-the-largest-digital-bank-RUS/>

³ OZON: OZON раскрывает финансовые результаты за 4кв.2024г. и 2024г. , URL: https://ir.ozon.com/ru/sth/ozon-obyavlyaet-finansovye-rezultaty-za-chetvertyy-kvartal-2024-goda-i-2024-god-2e5989f0?_rr=1&abt_att=1&origin_referer=yandex.ru

⁴ Ведомости: «Самолет» запустит собственный цифровой банк URL: <https://www.vedomosti.ru/press-releases/2022/10/18/samolet-zapustit-sobstvennii-tsifrovoy-bank>

Где искать высокую доходность на срочном рынке

Наш канал в Telegram

Самое важное о финансах, инвестициях, бизнесе и технологиях

Подписаться

Новости 04.03.2025

Четверть взрослых россиян страдают ожирением

Forbes Life

Jay-Z подал в суд на обвинившую его в изнасиловании женщину

На что способен флагман Samsung Galaxy S25 Ultra

30 самых дорогих компаний Рунета — 2023. Рейтинг Forbes

Рейтинг О рейтинге 2023 Поиск

№	Название	Стоимость	Руководитель
1.	Яндекс Яндекс IT-холдинг	10.2 млрд \$	Артем Савиновский
2.	WB РВБ (Wildberries) Маркетплейс	9.8 млрд \$	Татьяна Бакалчук
3.	ozon Ozon Маркетплейс	5.2 млрд \$	коллегиальный орган управления
4.	1С 1С Разработчик ПО	2.5 млрд \$	Борис Нуралиев
5.	касперский Лаборатория Касперского Кибербезопасность	2.4 млрд \$	Евгений Касперский
6.	Авито Авито Интернет-сервис	2.4 млрд \$	Владимир Правдивый
7.	Контур СКБ Контур Разработчик ПО	1.8 млрд \$	Михаил Сродных
8.	VK VK IT-холдинг	1.6 млрд \$	Владимир Кириенко

SHIFT

КВАРТИРЫ С ТЕРРАСАМИ И ПОТОЛКАМИ 3,6 М

Рис 2. В последние годы появились свои банки у IT-холдинга Яндекс, маркетплейсов OZON и Wildberries, СКБ Контур

Источник: Журнал Forbes <https://www.forbes.ru/tekhnologii/531173-30-samyh-dorogih-kompanij-runeta-2025-rejting-forbes>

Возможно, в ближайшие годы мы увидим ещё нескольких «чёрных лебедей» для традиционного банковского рынка, которые начнут агрессивно теснить старожилов. Всё это приводит к потере классическими банками громадных финансовых рынков. И если они не будут поспевать в своём инновационном развитии, то при такой стремительной потере рыночных ниш дерзкие новички вытеснят «аналоговые» банки на обочину истории.

Сегодня банковская система РФ испытывает беспрецедентное санкционное давление. При переходе на мирные рельсы появятся проблемы с кредитами у оборонных предприятий, сверх меры закредитовано население, кризис у девелоперов, резкий спад продаж на автомобильном рынке, а значит возможно значительное количество дефолтов в самых разных отраслях.

Именно в такой кризисной ситуации и появляется возможность кардинального перекаривания финансового рынка. Новички финансовой индустрии или ныне даже её аутсайдеры, стартовав в первых рядах небанков и финтех-проектов или более того оснащённые самой современной платформой Банк-как-Сервис, могут на громадной скорости ворваться в лидирующую группу российских банков. Благоприятствует тому абсолютно всё.¹

Большим плюсом для России является и то обстоятельство, что у нас в стране есть разработчики платформ Банк-как-Сервис, которые имеют успешный опыт работы за рубежом. Есть и впечатляющие, но, правда, пока ещё единичные примеры российских банков, работающих по этой технологии. Но рынок не резиновый. Он достаточно быстро будет насыщаться финансовыми организациями, работающими по этому стандарту. «Промедление... смерти подобно», - как писал классик русской революции.

Что такое «Банк-как-сервис» (Banking-as-a-Service, BaaS)

Банк-как-сервис (BaaS) — это финансовая модель, которая позволяет банкам предоставлять небанковским организациям банковские продукты через интерфейсы прикладного программирования (API)². Появившись в 1990-х годах, BaaS начался с традиционных розничных торговцев, сотрудничающих с банками для предоставления ими финансовых продуктов, тем самым превращая банковский ландшафт в совместную экосистему. Появление Интернета и цифровых технологий с тех пор ускорило

¹ Деловой Квартал: Малые банки сегодня способны заменить динозавров. Как модернизировать банковскую систему <https://www.dk.ru/news/237183843>

² PWC: Tech translated: Banking as a Service, URL: <https://www.pwc.com/gx/en/issues/technology/baas-banking-as-a-service.html> (дата обращения 2 марта 2025)

продвижение модели BaaS, сделав её основным инновационным подходом к началу 2020-х годов, при значительном участии как традиционных банков, так и финтех-компаний.

Быстрое развитие BaaS было обусловлено технологическими инновациями, такими как облачные вычисления и искусственный интеллект, которые повысили операционную эффективность и качество обслуживания клиентов. Кроме того, растущий интерес к встроенным финансам, когда компании интегрируют финансовые услуги в свои существующие сервисы, раскрыл потенциал BaaS, обеспечив прямой бесшовный доступ клиентов этих компаний к банковскому обслуживанию.

Платформы BaaS обеспечили экосистемам вычислительную эффективность (снизив транзакционные издержки на порядки), масштабируемость и гибкость. Перенос операций в финансовую экосистему, небанковские компании могут сократить расходы на инфраструктуру и расширить спектр своих услуг, тем самым резко усилив свои конкурентные позиции. Это позволяет банкам с более низкими издержками обрабатывать данные в реальном времени, адаптируясь к меняющейся динамике рынка и потребностям клиентов.

По нашему убеждению, существует принципиальное различие между понятиями *цифровой банк* и *необанк*. К цифровым банкам¹ мы относим, в первую очередь, банки, которые построены на BaaS-архитектуре, в то время как необанки – это, скорее, более удачные с точки зрения пользовательского опыта банковские приложения, которые могут быть подключены к архаичной банковской архитектуре. Полноценный цифровой банк может размещать на своей платформе неограниченное число различных необанков, создаваемых в том числе сторонними командами финтех-разработчиков.

Banking-as-a-Service в США

Концепция Banking-as-a-Service (BaaS) значительно изменилась с момента её создания в 1990-х годах. Первоначально крупные розничные торговцы, такие как Tesco и Sainsbury's, начали сотрудничать с банками, предлагая финансовые продукты и закладывая основу для совместной финансовой экосистемы. По мере развития технологий традиционные банковские практики претерпели трансформационные изменения, в частности, благодаря внедрению систем электронных переводов средств (EFT) ещё в конце 1970-х годов, которые позволили клиентам проводить транзакции без физических наличных денег. Это нововведение сыграло ключевую роль в подготовке почвы для последующей революции онлайн-банкинга.

1990-е годы стали поворотным десятилетием с появлением Интернета и Всемирной паутины, которые заложили основу для электронной коммерции и онлайн-банкинга. В эту эпоху появились крупные интернет-ритейлеры Amazon и eBay, а также такие технологические новации, как первое текстовое сообщение SMS и создание PayPal, что обеспечило внедрение электронных денежных переводов.

С приближением 21-го века такие технологические достижения, как рост Интернета и цифровых платформ, еще больше продвинули модель BaaS. Растущий интерес нефинансовых брендов к внедрению финансовых продуктов в свои клиентские сервисы содействовал изменению динамики рынка, способствуя более комплексному подходу к банковским услугам. К началу 2020-х годов BaaS превращается в перспективную инновационную услугу, и банки напрямую сотрудничают с поставщиками технологий для качественного улучшения своих услуг.

Ожидается, что в 2025 году в США 30% банков² с активами более 1 млрд долларов внедрят решения BaaS, что отражает растущую популярность модели в финансовом секторе. По прогнозам, к 2030 году объем рынка BaaS может достичь 75 миллиардов долларов, а среднегодовой темп роста (CAGR) составит более 16%³. Эта траектория роста обусловлена растущим спросом на цифровые банковские решения, поскольку компании все чаще будут использовать BaaS для улучшения взаимодействия с клиентами.

Искусственный интеллект и математический инструментарий

Искусственный интеллект (ИИ), машинное обучение и математические модели начали переопределять банковский ландшафт. Эти технологии позволяют проводить анализ данных в реальном времени, улучшая возможности управления рисками и обнаружения мошенничества. Алгоритмы ИИ помогают создавать персонализированный клиентский опыт с помощью индивидуальных рекомендаций и чат-ботов, которые оптимизируют обслуживание клиентов.

Экосистема, работающая на ИИ, анализирует поведение клиентов, предпочтения и риски, чтобы предложить персонализированные финансовые продукты и услуги. ИИ интегрирован в BaaS-платформу. Интеграция ИИ выходит за рамки традиционного банковского бизнеса, способствуя инновациям, поддерживающим совместное финансирование, и оптимизирует бизнес-процессы во всем секторе.

Наиболее современные математические модели позволяют производить достаточно сложные расчеты в реальном времени, например, решать задачу управления ликвидностью в мультивалютных

¹ MasterCard: The opportunities of Banking as a Service, URL: <https://www.mastercardservices.com/en/advisors/sustainability-financial-inclusion-consulting/insights/opportunities-banking-service>

² HES FinTech: Banking-as-a-Service Explained: BaaS Model Examples, URL: <https://hesfintech.com/blog/banking-as-a-service-explained/>

³ The Financial Revolutionist: The Evolution of Baas, URL: <https://thefr.com/news/the-evolution-of-baas>

финансовых платформах. В частности, в 2023 году авторы российской BaaS-платформы «Цифровая динамика» разработали подобную математическую модель¹.

Таким образом, современный математический инструментарий, будучи встроенным в BaaS-платформы, может качественно изменить характер решаемых задач, таких как токенизация активов, расчеты через цифровые финансовые активы, построение и анализ бартерных цепочек и цепочек взаимозачетов, а также ряд других аналитических задач.

Пропадет ли интерес к BaaS-системам у рынка?

Поскольку различные формы BaaS-решений существуют уже достаточно давно, есть соблазн объявить BaaS-решения неким уже пройденным этапом. Однако, мы считаем, что BaaS-системы все эти годы продолжают эволюционировать и стремительно развиваться.

Основная, на наш взгляд, проблема многих BaaS-решений на зарубежном и российском рынке – это часто их ограниченность исключительно платежными функциями. Во многих случаях BaaS-решение состоит только из предоставления услуг по карточному эквайрингу или решению задач массовых выписок. С этой задачей за десятки лет научились справляться многие.

Однако, указанные выше функции – это лишь небольшая часть того, что может и должно предоставлять полноценное BaaS-решение. По нашему мнению, BaaS-платформа вообще закрывает весь продуктовый функционал современного банка и даже более того. Об этом пойдет речь ниже. Если банк построен на базе такой платформы, то он может быть отнесен к полноценному цифровому банку, а на базе платформы цифрового банка уже можно строить интегрированные банковские решения.

Как создается полноценная BaaS-платформа?

Подобные разработки не появляются на пустом месте. Конструировать такие проекты всегда принципиально сложно, поэтому есть лишь небольшое число команд, которые могут разрабатывать BaaS-платформы. Просто так издать в банке распоряжение о том, чтобы начали разработку своей системы, невозможно, так как нужно собрать полноценную разностороннюю команду, научить её общаться на одном языке, сформулировать требования, фактически вывести людей за пределы операционной деятельности банка и «из зоны комфорта».

Наиболее удачные продукты создаются командами, которые работали в банковском (и вообще финансовом) секторе многие десятилетия, имеют академическую подготовку в математическом моделировании. Причем, зачастую наиболее удачный продукт получается лишь с 3-й, а то и с 5-й версии, каждая из которых с рыночной её апробацией занимает до 5 лет, а также следует учесть весь накопленный ранее практический опыт. Важна бывает разносторонность проектов, а не только сосредоточенность на банковских операциях. Например, опыт в сфере трейдинга, токенизации, исламских финансов чрезвычайно важен, особенно международный опыт. Очень часто узкая специализация разработчиков платформ только на российском рынке мешает в дальнейшем правильно масштабировать проекты на другие страны при зарубежной экспансии.

Важно отметить, что при проектировании BaaS-платформ необходимо закладывать требования и ограничения регуляторов, которые существуют для финансовых приложений в разных странах. В частности, в РФ важно соответствие требованиям Центрального банка в части идентификации клиентов, финансового мониторинга, способа удостоверения транзакций, а также соответствия специальному профилю безопасности системы, о чем речь будет идти ниже в специальном разделе.

Решение Банк-как-Сервис предполагает формат платформенного b2b-бизнеса, который направлен на две группы клиентов одновременно:

- с одной стороны, *традиционные банк(и)*, для которых подключение к решению BaaS позволяет получить современную высокотехнологичную API-платформу и расширить перечень своих сервисов,
- а с другой стороны, *SaaS-компании и IT-разработчики*, которые смогут интегрировать банковские сервисы в свою продуктовую линейку.

Основная мотивация для экосистем/банков/маркетплейсов – быстрый переход, буквально за два-три месяца, на высокотехнологичные рельсы без необходимости многолетней in-house разработки ПО. Мотивация для SaaS/IT-компаний – это расширение спектра сервисов своих клиентов за счет бесшовной интеграции с банками, в том числе и из второй сотни, ранее не имевших таких возможностей.

Для заказчиков предлагаются две основные категории сервисов:

1. **Работа экосистемы на BaaS-платформе с банковскими сервисами через API** - встраиваемый и интеграционный банкинг с большим набором уже готовых базовых банковских сервисов;
2. **Специализированные уникальные сервисы BaaS-платформы** - например, трансграничные операции на базе расчетов между клиентами с использованием бартерных цепочек и токенизация произвольных материальных и нематериальных активов, например, операций с драгметаллами, киловатт-часами, квадратными метрами и любыми другими.

Также в требованиях к BaaS-системе есть ряд дополнительных условий, а именно:

¹ Frolov, V.N., Vatolin, A.A. & Romanchuk, A.P. Asset Tokenization and Related Problems. *Proc. Steklov Inst. Math.* **323** (Suppl 1), S98–S112 (2023)

- высокая степень автоматизации операций, позволяющая кардинально (на порядки) минимизировать количество персонала в финансовом учреждении и, как следствие, получить существенно более низкую себестоимость операций;
- возможность непрерывного ведения бизнеса с установкой всех обновлений без перерыва на обслуживание клиентов;
- высокая скорость разработки системы и быстрая доставка разработанных продуктов до клиента с прохождением всех этапов контроля качества, норм информационной безопасности;
- возможность разворачивать новую BaaS-инсталляцию за кратчайшие сроки (2–3 месяца).

Описание функционала BaaS-платформы

BaaS-платформа представляет собой гибкий конструктор для создания различных банковских сервисов, начиная от базовых продуктов, таких как *рублевое РКО* и заканчивая более экзотическими сервисами, такими как *токенизация активов или, скажем, оборот драгметаллов*.

На старте платформа должна иметь полную линейку базовых банковских продуктов, таких как:

- *Платежи в различных их видах и конфигурациях* – по произвольным реквизитам, через Систему быстрых Платежей (СБП), интеграции с разными поставщиками платежных решений;
- *Базовые продукты для физических лиц*, такие как ведение счетов, интеграция с карточными системами, а также для юридических лиц, такие как РКО, тарификация, картотека, блокировки ФНС, выписки по счетам и пр;
- *Депозиты для физических и юридических лиц* – справочник продуктов, заявки на открытие/закрытие, различные заявления на изменения условий (например, распоряжение процентами, капитализацией);
- *Кредитные продукты для физических и юридических лиц* – справочники, кредитные заявки, ведение договоров;
- Для юридических лиц – *работа с приемом платежей* через СБП, QR-кодами, регистрация торговых точек, интеграция эквайринговых решений;
- *Работа с валютными счетами*, ведение ВЭД-контрактов, конвертации валют, валютные платежи;
- *Различный банковский документооборот* – зарплатные ведомости, заявления для банковских карт и пр.
- *Интеграция с торговыми системами*.

Де-факто, решение Banking-as-a-Service представляется собой API-платформу. И все вышеуказанные продукты могут быть встроены в различные информационные системы, реализуя концепцию встраиваемого и интеграционного банкинга: например, облачные CRM, агрегаторы депозитов и кредитных заявок, различные бухгалтерии и системы управления персональными финансами.

BaaS-платформа может иметь ряд уникальных продуктов, которые помогут подключившимся банкам и экосистемам сделать их существенно более продвинутыми:

- *Использование автоматизированных цепочек взаимозачетов* для схлопывания дебиторских задолженностей юридических лиц без использования денежных средств (на клиентах большого числа банков это будет работать особенно эффективно);
- *Использование бартерных цепочек* в автоматизированных расчетах для полного или частичного замещения транзакций с использованием денежных средств;
- *Токенизация различных активов* (например, ресурсов, таких как зерно, минеральные удобрения, полезные ископаемые, металлы, квадратные метры жилья, киловатт-часы электроэнергии и пр.);
- *Экосистема драгметаллов* (покупка и продажа драгметаллов, конвертация их в ювелирные изделия, прием лома драгметаллов с конвертацией в электронный вид и иные функции);
- *Система трансграничных платежей*, построенная по принципу использования базовой расчетной валюты (например, золота) или по принципу взаимозачета и учета встречных обязательств.

Отдельно в экосистеме на BaaS-платформе можно дополнительно запустить независимые проекты, интегрированные с банками, подключенными к ней: операторы электронных кошельков, платежные системы, брокеры для работы с драгметаллами, риелторы, операторы маркетплейсов и пр.

Стоит отметить один важный момент. Поскольку решение BaaS является по сути API-платформой, то её интеграции работают в две стороны – на одной стороне множество банков, на другой стороне – различные SaaS-провайдеры или разработчики IT-решений и продуктов для финансового сектора. Оператор BaaS-платформы публикует *открытый API* и приглашает всех разработчиков к сотрудничеству. Таким образом, в полной мере реализуется реальный экосистемный подход, основанный на использовании открытых API.

Архитектура цифрового банка на BaaS-платформе

Мы считаем, что один из базовых принципов при проектировании BaaS-платформы - это принцип *разделения ответственности между банковскими системами*. В частности, мы предлагаем выносить

за пределы АБС (автоматизированной банковской системы) все задачи, связанные с разработкой банковских продуктов, оставляя для АБС лишь базовые функции, такие как корректное ведение учета, формирование отчетности в соответствии с требованием регуляторов и некоторые другие, связанные с исполнением прежде всего регуляторных требований.

Такой подход позволяет существенно упростить разработку и тестирование банковских продуктов, поскольку АБС, как правило, редко соответствуют современным принципам ведения разработок, таким как наличие автоматических тестов, версионность, CI/CD (непрерывная интеграция и непрерывное развертывание программного обеспечения в процессе разработки), не говоря уже о невозможности использования современных языков программирования, а также часто жесткой привязки к иностранным коммерческим продуктам (например, Oracle).

В итоге мы получаем следующую формулу идеального архитектурного устройства современного цифрового банка:

$$\text{Банк/НКО} = [\text{Учет/Отчетность}] + [\text{Сервисы/Продукты}] + [\text{UI/UX}]$$

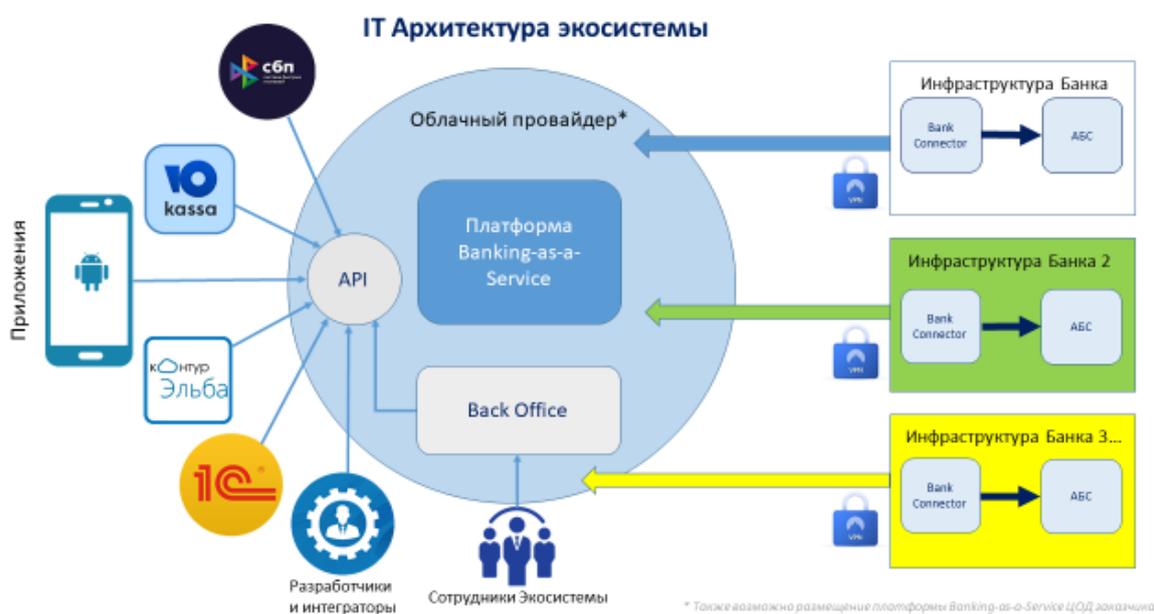


Рис 3. IT-архитектура финансовой экосистемы на базе BaaS-платформы

Всё, что относится к Учету и Отчетности, мы предлагаем реализовывать в АБС, всё, что касается Сервисов и Продуктов, должно быть на стороне BaaS-платформы либо микросервисов, непосредственно к ней подключенных. А третий уровень – UI/UX – это разнообразные мобильные, веб, чат-боты или иные приложения, которые уже реализуют огромное разнообразие пользовательских интерфейсов, подключаясь к API-слою BaaS-платформы.

Конечно же у банков есть и другие системы, такие как CRM, карточный процессинг (собственный или сторонний), СБП-шлюз, платежные системы и пр., но все они носят уже вспомогательный характер для банков и интегрируются либо с BaaS-платформой, либо, в крайнем случае, с АБС.

Фактически, каждый подключенный к BaaS-платформе банк автоматически становится *высокотехнологичным цифровым банком* с API. При наличии собственной или привлеченной команды банк или оператор экосистемы сможет вести разработку собственной продуктовой линейки на базе функционала Платформы и *сосредоточиться именно на конечных продуктах*, поскольку BaaS-платформа берет на себя все нюансы автоматизации и интеграции внутренних систем банка.

Усиление информационной безопасности

Проблема информационной безопасности и защиты от взлома для BaaS-платформы является ключевой, поскольку любые проблемы в API могут превратиться в многомиллиардные убытки для финансовых компаний. До сих пор именно проблемы безопасности являются основной проблемой запуска BaaS-решений, да и в целом любых других банковских решений¹.

¹ Портал CNEW: «Солар» назвал топ-5 критических уязвимостей банковских приложений, URL: https://banks.cnews.ru/news/line/2025-02-27_solar_nazval_top-5_kriticheskikh (дата обращения 2 марта 2025)

В РФ в соответствии с рекомендациями ЦБ существует несколько требований к информационной безопасности банков, например, таких как соответствие уровню доверия (ОУД4), требования к проведению постоянных тестов на проникновение и иных регламентов. Для компаний-разработчиков ПО для банков и иных финансовых организаций крайне важно не только обеспечить соответствие данным требованиям, но и реализовать такой процесс разработки своего ПО, чтобы он исключал возможность появления уязвимостей, критических ошибок или же вмешательства сторонних лиц в процесс разработки.

Иными словами, очень важно выстроить процесс разработки так, чтобы он соответствовал критериям безопасной разработки. Компании-разработчики, как правило, строят безопасный конвейер - он включает в себя разграничения доступа к исходному коду, безопасной сборки и доставки ПО, когда исключается вмешательство извне, происходит проверка источника любых изменений в создаваемом продукте. В процессе сборки происходит многоуровневое автоматическое тестирование, позволяющее выявить как возможные ошибки, так и потенциальные проблемы безопасности. Готовое ПО проходит обязательный автоматический анализ на предмет уязвимостей (статический анализ, анализ библиотек, динамический анализ уязвимостей), автоматическое тестирование на исполнение требований ОУД4, и только после этого собранные дистрибутивы могут быть отгружены на тестовый стенд или заказчику.

Дополнительно, после развертывания ПО на стенде проводится периодический динамический анализ ПО различными сканерами либо силами отдела информационной безопасности, либо сторонними аудиторами или даже специально привлекаемыми для этого командами так называемых «белых хакеров».

Итоговая цель таких мероприятий – это многоуровневая проверка, исключающая появление ошибок и уязвимостей в ПО всей экосистемы, построенной на BaaS-платформе.

Заключение и выводы

За последние два десятилетия в мире уже есть все предпосылки для создания принципиально новых банковских институтов и платёжных решений. С появлением Интернета, смартфонов и повсеместной высокоскоростной мобильной связи банки и самые разные платёжные инструменты стали доступны, что называется, «на ладони», а пользователям для проведения транзакций достаточно буквально «одного пальца» и наличия денежных средств на счету в цифровом банке.

На рынках появились неоланки и полноценные *цифровые банки*, без классических офисов, без лишней бюрократии и наследия старого архаичного ПО, гибкие, современные, открытые для новейших технологий.

Кроме чисто банковских организаций, на долю финансового рынка претендуют игроки и из других секторов экономики: IT-бизнеса, телеком-индустрии, электронной коммерции и пр., которые могут стартовать в кратчайшие сроки за счёт использования платформы Банк-как-Сервис в качестве основного движка для автоматизации транзакций и подключения любых финтех-решений через API к цифровому банку. Те компании, которые не торопятся внедрять самые передовые решения в цифровом финансовом мире, рискуют навсегда потерять свои некогда устойчивые позиции и причём уже в ближайшее время.

*Фролов Владимир Николаевич,
доктор эконом. наук, профессор научный руководитель
АО «Цифровая Динамика», г. Екатеринбург email: frolov@anr.ru*

*Романчук Алексей Петрович,
генеральный директор
АО «Цифровая Динамика», г. Екатеринбург
email: a@dw-dynamics.ru*

*Дорофеев Андрей Викторович,
директор по развитию
АО «Цифровая Динамика», г. Екатеринбург
e-mail: avd@dw-dynamics.ru*

Ключевые слова

Банк-как-Сервис, финтех, маркетплейс, экосистема, трансграничные платежи, платёжные системы, неоланк, цифровой банк, рынок драгметаллов, международные финансы, Интерфейс прикладного программирования - API, Автоматизированная банковская система - АБС, Дистанционное банковское обслуживание - ДБО, CI/CD - непрерывная интеграция и непрерывное развертывание, оценка доверия, токенизация, искусственный интеллект (ИИ), е-коммерция.

Vladimir Frolov, Alexey Romanchuk, Andrey Dorofeev. Bank-as-a-Service is a parallel reality of financial ecosystems.

Keywords

Bank-as-a-Service (BaaS), fintech, marketplace, ecosystem, cross-border payments, payment systems, neobank, digital bank, precious metals market, international finance, Application Programming Interface (API), Automated Banking System (ABS), Remote Banking, continuous integration and continuous deployment (CI/CD), continuous delivery, evaluation trust, tokenization, artificial intelligence (AI), e-commerce.

DOI: 10.34706/DE-2025-01-11

JEL classification G21 – Банки», депозитные организации, микрофинансовые организации.

Abstract

The article examines the latest Bank-as-a-Service (Banking-as-a-Service, abbreviated BaaS) technology in the financial industry, the history of payment solutions, the prerequisites for the emergence of BaaS, the application features, the advantages of using this solution for banks and non-banking companies, as well as challenges for the conservative banking community.

Памятка для авторов публикаций в журнале «Цифровая экономика»

В нашем журнале выполняются все требования Diamond-ОА, включая отсутствие платы как со стороны авторов, так и со стороны читателей, рецензирование, а также проверка на плагиат и избыточное самоцитирование. Авторские права на опубликованные статьи остаются за авторами.

В журнале нет штатных сотрудников, все работы, включая проверку на плагиат, рецензирование, работу корректора и форматирование, выполняются группой единомышленников на общественных началах, а потому мы рассчитываем на такое же отношение к своим правам и обязанностям со стороны авторов. Материалы, опубликованные ранее (полностью или в значительной своей части) в других изданиях, не принимаются. Мы очень надеемся, что предполагаемые авторы избавят нас от работы с такими текстами.

Первое, что предлагается автору, желающему опубликовать статью в нашем журнале, – это зарегистрироваться в качестве потенциального автора и самому разместить предлагаемый к публикации текст на сайте журнала в отведенном для этого разделе (научные статьи, мнения, обзоры, рецензии, переводы). Тем самым автор принимает условия журнала и дает добро на публикацию своей статьи в журнале после прохождения всех предусмотренных процедур. Этот принцип мы стараемся соблюдать, но допускаем размещение присланных статей редактором, поскольку по техническим причинам (атаки ботов и т.п.).

Также рекомендуется присылать текст в точности в том формате, как он будет размещен в журнале, то есть 9 кегль Arial основной текст и 8 кегль в таблицах, рисунки и таблицы должны точно вписываться в страницы, не переползать на следующую страницу и не оставлять белое поле перед. Желательно использовать готовую форму и вписывать в нее. Правильно оформленные статьи будут иметь приоритет, поскольку облегчают нашу работу. Мы умеем это ценить

Полная готовность научной статьи к публикации означает ее соответствие принятому в журнале стандарту, включая правильное оформление списка литературы и ссылок, полные сведения об авторах, индексы JEL и УДК, аннотацию и ключевые слова на русском и английском, редактируемые формулы (набранные Word и в нем же редактируемые), ручную нумерацию разделов, рисунков и таблиц. Если нумерация автоматическая, она может сбиться при вставке статьи в общий блок.

Заголовок не должен быть длинным. Иначе в колонтитуле будет бессмыслица. Не надо набирать заголовок большими буквами. Надо использовать опцию «все прописные». Это важно!

В списке литературы научные статьи упорядочиваются по алфавиту, причем сначала идут русскоязычные публикации, потом англоязычные и пр. Это нужно, чтобы не возникло путаницы при формировании транслитерации кириллических статей. Источники данных, нормативные и методические материалы идут отдельным списком. Ссылки на интернет-ресурсы, газетные публикации и т.д. желательно давать в сносках. Ссылки на научные публикации должны быть даны в формате [Автор, 2023]. При необходимости к году может быть добавлена латинская буква 2023a, 2023b.

Публикация статьи означает получение ей метаданных, включая DOI, номер выпуска, страницы. Выпуск журнала делается в формате pdf, причем в таком виде, что его сразу можно отдать в типографию и сделать твердую (бумажную) копию, если кто-то из авторов хочет ее иметь для себя. Бумажная версия выпуска имеет статус буклета, печатается за счет автора (заказчика) и в количестве, определенном заказчиком.

Статьи, размещенные авторами на сайте журнала, доступны читателям немедленно, еще до того, как прошли рецензирование. Они не считаются опубликованными до прохождения рецензирования и технических процедур. Но самим фактом размещения и предварительной регистрации человек разрешает это опубликовать, отпадает необходимость в письменном договоре. Если автор присылает статью в журнал и просит ее разместить, он нарушает стандартную процедуру и может создать нам сложности в будущем. Старайтесь следовать правилам и не создавать нам проблем!