УДК 004.046
DOI: 10.18799/29495407/2025/4/107
Шифр специальности ВАК: 2.3.1
Научная статья

Концепция формализации и программной реализации ИТ-задач: анализ, модель, алгоритмизация, инженерия

Е.А. Кочегурова✉, О.Н. Ефремова, О.Н. Имас, А.И. Шерстнева, С.Г. Цапко

Национальный исследовательский Томский политехнический университет, Россия, г. Томск

✉kocheg@tpu.ru

Аннотация. Целью данной работы является развитие концепции и создание методологической основы перехода между ИТ-вызовами и их практической реализацией. Ключевым элементом предлагаемого подхода является концепция сквозной формализации ИТ-задач, устанавливающей взаимосвязь между этапами: постановка проблемы→математическая модель→вычислительное решение→промышленная реализация. Методологической основой такого перехода служит системное применение аппарата фундаментальной и прикладной математики, обеспечивающее научную обоснованность выбранных инженерных решений. Кроме того, уделяется внимание этапу верификации полученных моделей и оценке их готовности к программной реализации в условиях промышленной эксплуатации.

Ключевые слова: ИТ-задача, формализация и абстракция, фундаментальная и прикладная математика, ИТ-реализация

Благодарности: Работа выполнена при поддержке гранта Автономной некоммерческой организации «Аналитический центр при Правительстве Российской Федерации» («Топ-ИТ») № 70-2025-000841.

Для цитирования: Концепция формализации и программной реализации ИТ-задач: анализ, модель, алгоритмизация, инженерия / Е.А. Кочегурова, О.Н. Ефремова, О.Н. Имас, А.И. Шерстнева, С.Г. Цапко // Известия Томского политехнического университета. Промышленная кибернетика. – 2025. – Т. 3. – № 4. – С. 21–28. DOI: 10.18799/29495407/2025/4/107

UDC 004.046
DOI: 10.18799/29495407/2025/4/107
Scientific paper

Formalization concept and software implementation of IT problems: analysis, model, algorithmization, engineering

E.A. Kochegurova✉, O.N. Efremova, O.N. Imas, A.I. Sherstneva, S.G. Tsapko

National research Tomsk Polytechnic University, Tomsk, Russian Federation

✉kocheg@tpu.ru

Abstract. The aim of the paper is to develop a concept and create a methodological basis for transition between IT challenges and their practical implementation. A key element of the proposed approach is the concept of end-to-end formalization of IT tasks, establishing a relationship between the following stages: problem statement→mathematical model→computational solution→industrial implementation. The methodological basis for this transition is the systematic application of fundamental and applied mathematics, ensuring the scientific validity of the engineering decisions made. Furthermore, attention is paid to the verification stage of the resulting models and assessment of their readiness for software implementation in industrial conditions.

Keywords: IT problem, formalization and abstraction, fundamental and applied mathematics, IT implementation

Acknowledgements: This work was supported by the grant from the Autonomous Non-Commercial Organization "Analytical Center under the Government of the Russian Federation" ("Top-IT") no. 70-2025-000841.

For citation: Kochegurova E.A., Efremova O.N., Imas O.N., Sherstneva A.I., Tsapko S.G. Formalization concept and software implementation of it problems: analysis, model, algorithmization, engineering. *Bulletin of the Tomsk Polytechnic University. Industrial Cybernetics*, 2025, vol. 3, no. 4, pp. 21–28. DOI: 10.18799/29495407/2025/4/107

Обоснование актуальности формализации

В условиях цифровой трансформации возникает потребность в точных и верифицируемых методах описания сложных систем. Классическая математика предоставляет универсальный формальный аппарат для описания характеристик процессов в любой области промышленности и бизнеса. Именно математика обеспечивает точность и универсальность математических формулировок, используя формальный аппарат закономерностей, моделей и формул.

Беспрецедентный рост количества и сложности задач ИТ-отрасли требует замены интуитивных решений на формальные методы. В этих условиях необходим унифицированный подход, позволяющий декомпозировать любую прикладную задачу от содержательной постановки до конечной программной реализации.

Поэтому создание методологических основ и практического руководства, а также выявление самих этапов перехода представляется задачей актуальной для всех прикладных областей и для ИТ-отрасли особенно.

Ключевая идея концепции основана на логической сути перехода от содержательной ИТ-задачи к ее практической реализации в виде ИТ-системы. При этом исходная ИТ-задача порождает задачи для фундаментальной и прикладной математики, а алгоритмическое и программное решения являются ядром для ИТ-системы. Сам процесс перехода может быть представлен схемой (рис. 1) и является методологической основой для такого перехода.

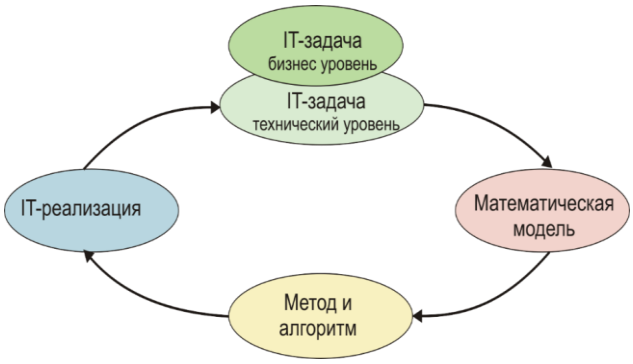

Рис. 1. *Схема этапов перехода от ИТ-задачи к ИТ-реализации*
Fig. 1. *Diagram of the stages of transition from an IT task to IT implementation*

Процесс перехода может быть разделен на четыре ключевых этапа:

1. Постановка задачи в предметной области (ИТ-задача).
2. Задача классической математики (формализация и абстракция).
3. Решение методами прикладной математики (выбор и адаптация алгоритмов).
4. ИТ-реализация (программная инженерия и внедрение).

Постановка задачи в предметной области (ИТ-задача)

Начальный этап жизненного цикла решения является важным, поскольку формирует основу для всех последующих этапов. Цель этого этапа – трансформировать содержательное описание задачи некоторой предметной области в четко сформулированную ИТ-задачу для решения средствами информационных технологий.

Успешность этапа определяет решение следующих основных задач:

1. Достижение однозначного понимания содержания задачи всеми участниками процесса от заказчика до разработчика. Необходимо установить однозначную интерпретацию сути и ожидаемых результатов задачи. Это позволяет минимизировать риски неверной трактовки требований на дальнейших этапах.
2. Структурная декомпозиция. Требуется выделить структурные компоненты задачи. Этот процесс позволит определить ключевые функции, модули, данные и сущности. Также на этом этапе нужно выявить взаимосвязи между компонентами и оценить сложность реализации каждого из них.
3. Первичная формализация задачи. Это облегчает понимание и интерпретацию задачи различными специалистами [1].

По возможности нужно выполнить первичную формализацию и описать задачу в терминах

- бизнес-логики (перечень процессов и правил, по которым система работает);
- пользовательских требований (функциональные и нефункциональные потребности пользователя и заказчика);
- технологических ограничений.

Результатом этапа является четкое описание задачи, сформулированное в виде технического задания или спецификации требований [2, 3]. Эти документы необходимы для последующей математической формализации задачи, выбора методов аналитического или численного ее решения и разработки алгоритмов.

При этом основное внимание уделяется именно пониманию сути задачи, а не поиску возможных решений или путей её реализации. Это может быть задача повышения эффективности, автоматизации процесса, анализа данных или создания нового функционала.

Задача классической математики (формализация и абстракция)

Цель этапа математического моделирования и формализации – перевести ИТ-задачу на язык строгих математических категорий и понятий. Для этого осуществляется сопоставление структурных элементов задачи, выделенных на первом этапе, с

соответствующими математическими категориями. Требуется абстрагироваться от конкретных технологий и предметной области и выбрать подходящий раздел фундаментальной математики (логика, теория графов, оптимизация и др.) для выделенных структурных компонентов ИТ-задачи [4].

Этот этап формализации является ключевым, поскольку позволяет применить мощный инструментарий фундаментальной математики для анализа и решения задачи и включает два шага.

1. Идентификация типа задачи и выбор математического аппарата. Необходимо сопоставить элементы задачи с *конкретным математическим аппаратом* и записать в терминах принятой математической нотации, т. е. получить формальную математическую модель. При этом существуют только общие рекомендации по соответствию типа задачи разделам математики. Некоторые из них приведены в табл. 1.

Таблица 1. *Соответствие между типом задачи и разделами фундаментальной математики*

Table 1. *Correspondence between the type of problem and the section of fundamental mathematics*

Тип задачи Task type	Типовые элементы задачи Typical elements of the task	Рекомендуемый раздел Recommended section
Структуры и связи (*структурно-реляционная*) Structures and relationships (*structural-relational*)	объекты, сети, иерархии objects, networks, hierarchies	Теория графов, Теория множеств Graph Theory, Set Theory
Логика и состояния (*логико-алгоритмическая*) Logic and states (*logical-algorithmic*)	условия, переходы, правила валидации conditions, transitions, validation rules	Математическая логика, Теория автоматов Mathematical logic, Automata theory
Оптимизационная Optimization	поиск лучшего варианта searching for the best option	Математическое программирование, Дифференциальное исчисление Mathematical programming, Differential calculus
Данные и неопределенность (*вероятностно-статистическая*) Data and uncertainty (*probability-statistical*)	анализ, прогнозы, случайность analysis, forecasts, chance	Теория вероятностей, Статистика, Линейная алгебра Probability Theory, Statistics, Linear Algebra
Динамика и преобразования Dynamics and transformations	изменение во времени, движение change in time, movement	Дифференциальные уравнения, Линейная алгебра Differential equations, Linear algebra

Рекомендации по выбору раздела математики довольно субъективны, опираются на опыт пользователя и даже интуицию. Однако на основании систематических обзоров и рекомендаций рядом авторов [5, 6], можно руководствоваться объективной частотой использования математических разделов и основных областей их применения (рис. 2).

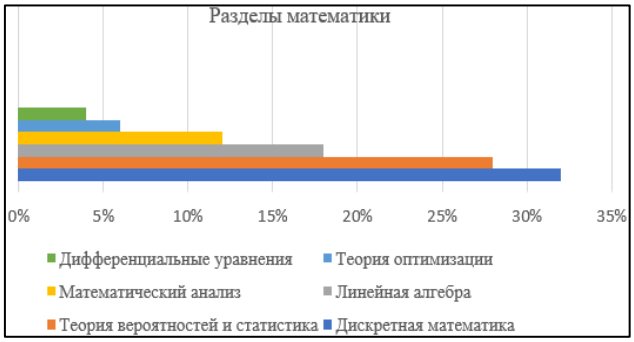

Рис. 2. *Частота использования разделов математики в ИТ-задачах*

Fig. 2. *Frequency of using sections of mathematics in IT tasks*

2. Построение формальной математической модели. На практике производится конкретизация абстракций – выбранный математический аппарат преобразуется в работоспособную модель системы. Если предыдущий шаг в целом определяет математический аппарат описания задачи, то текущий наполняет его конкретным содержанием с видом математических зависимостей, значений параметров и коэффициентов модели.

Например, если выбранный аппарат – математическое программирование, то на этапе формализации устанавливается:

- конкретный вид целевой функции на основании критерия оптимизации (затраты – минимизировать; производительность – максимизировать). Если стоит задача минимизации затрат, целевая функция принимает вид стоимостного выражения; если требуется максимизация производительности – она отражает метрики эффективности системы;
- ограничения, которые определяют условия функционирования системы. Ограничения выражаются через алгебраические выражения и неравенства, описывающие доступные ресурсы (например, память ≤64 ГБ, пропускная способность ≤1 Гбит/с), технические требования (время отклика <200 мс).

Таким образом, формирование математической модели включает определение целевой функции и набора ограничений, что позволяет представить систему в виде четкой математической задачи.

В табл. 2 приведены некоторые примеры конкретизации математического аппарата для задач ИТ.

Таблица 2. *Примеры соответствия между ИТ-задачами и разделами математики*

Table 2. *Examples of correspondence between IT task and a section of mathematics*

Тип задачи Task type	Примеры ИТ-задач Examples of IT tasks	Рекомендуемый раздел Recommended section
Задачи с линейными зависимостями, системами и преобразованиями Problems with linear dependencies, systems and transformations	• расчет итоговой стоимости заказа с учетом скидок и налогов Calculation of the final cost of the order taking into account discounts and taxes; • рекомендательная система на основе линейных моделей Recommender system based on linear models • изменение размеров/положения объектов в 2D-графике Changing the size/position of objects in 2D graphics	• линейная алгебра/linear algebra: • исходные данные – вектора/initial data – vectors; • преобразования – матрицы transformations – matrices; • условия и ограничения – системы линейных уравнений conditions and constraints – systems of linear equations
Задачи, связанные с динамикой и оптимизацией Problems related to dynamics and optimization	• нахождение оптимального размера партии данных для передачи при ограниченном времени Finding the optimal data batch size for transmission under limited time; • моделирование плавного движения или роста показателей modeling smooth movement or growth of indicators; • поиск экстремальных значений (макс. прибыль, мин. затраты) search for extreme values (max. profit, min. costs)	математическое программирование, дифференциальное исчисление, линейная алгебра mathematical programming, differential calculus, linear algebra
Задачи, связанные с устойчивостью, асимптотическим поведением и бесконечными процессами Problems related to stability, asymptotic behavior and infinite processes	• анализ производительности системы при экстремальных нагрузках analysis of system performance under extreme loads; • расчет сложных процентов или роста базы пользователей за длительный период calculating compound interest or user base growth over a long period; • анализ сходимости итерационных алгоритмов analysis of convergence of iterative algorithms	Пределы $\lim (n\to\infty)\ T(n)$, где n – число одновременных запросов; $T(n)$ – время отклика системы; $\lim (k\to\infty)\ x\_k=x^*$, где x_k – решение на k-й итерации; x^* – точное решение Limits $\lim (n\to\infty)\ T(n)$, where n is the number of simultaneous requests; $T(n)$ is the system response time; $\lim (k\to\infty)\ x\_k=x^*$ where x_k is the solution at the k-th iteration; x* is the exact solution
Задачи, связанные с геометрией объектов, положениями в пространстве и расстояниями Problems involving the geometry of objects, positions in space, and distances	• определение попадания точки (курсора) в область (кнопку) determining whether a point (cursor) enters an area (button); • расчет расстояния между объектами на карте; проверка столкновений в простой 2D-игре Calculating the distance between objects on the map; checking for collisions in a simple 2D game	аналитическая геометрия – объекты описываются уравнениями линий (окружность кнопки, прямая траектории), расстояния вычисляются по аналитическим формулам, проверка условий – подстановка координат в неравенства (определяющие область) analytical geometry – objects are described by line equations (the circle of a button, the line of a trajectory), distances are calculated using analytical formulas, condition checking involves substituting coordinates into inequalities (defining the domain)

Таким образом, результатом этого этапа становится законченная математическая задача, готовая к применению соответствующих методов решения. Модель сохраняет все существенные характеристики исходной IT-проблемы, но выраженные на строгом языке математики, что позволяет перейти к этапу аналитического, или численного решения.

Решение методами прикладной математики (выбор и адаптация алгоритмов) перенести

Цель данного этапа – получить точное или приближенное решение конкретной математической задачи, сформулированной на этапе математической формализации.

Этап включает три шага:

А. Выбор численного метода.

Б. Адаптацию метода.

В. Верификацию и валидацию численного метода.

А. После формализации требуется выбрать и/или адаптировать конкретный метод для решения полученной ранее математической задачи. Например, аналитический метод или численный метод. Выбранный метод реализует задачи соответствующего раздела математики: линейной алгебры, дифференциальных уравнений, динамического программирования и др. [7, 8].

Аналитический, или прямой, метод решения наиболее привлекателен, т. к. обеспечивает высокую точность и быстродействие, хорошую сходимость и устойчивость решения. Однако спектр

аналитических методов довольно ограничен, поскольку основан на точных формулах. А их существует небольшое количество. Например, при наличии формальной математической модели в виде нелинейного полиномиального уравнения аналитические методы его решения ограничены кубическим уравнением и не выше. А для нелинейных трансцендентных уравнений аналитические методы отсутствуют.

Численные методы чаще всего итерационные, т. е. решение в них является результатом последовательного улучшения начального приближения, выбор которого – также отдельная и нетривиальная задача. А отсюда особенности и недостатки многих численных методов: неизвестное заранее реальное быстродействие метода, необходимость оценки устойчивости и сходимости. Отдельная особенность – наличие методологической погрешности, которой нет в аналитических методах. Наряду с вычислительной погрешностью, которая присутствует в любых методах решения при использовании цифровой техники.

С понятием численного метода тесно связано понятие численного алгоритма – способа реализации метода в виде последовательности действий и шагов. Алгоритм – это реализация метода и основа для программной реализации [9].

При выборе численного метода необходимо руководствоваться не только корректностью метода (сходимость + устойчивость), но и такими характеристиками алгоритма, как точность, эффективность и сложность.

Мы не только выбираем подходящий алгоритм, но и оцениваем его вычислительную сложность, устойчивость и адекватность для нашей конкретной формализованной модели.

Б. Адаптация метода.

Следующий шаг после выбора численного метода – его адаптация к реальной задаче. Адаптация нужна для превращения абстрактного математического алгоритма в инструмент для решения конкретной инженерной или научной задачи.

Прежде всего, реальная задача требует больших объемов вычислений. Для управления вычислительными ресурсами могут быть использованы следующие пути адаптации:

- баланс между точностью и скоростью решения. Это самый очевидный способ адаптации численного алгоритма к реальной задаче. И между этими двумя показателями можно устанавливать паритет в зависимости от требований ИТ-задачи. В задачах реального времени критично быстродействие, сопоставимое с тактом опроса в системе (100 мс, 1 с и др.). И наоборот, при апостериорном оценивании оптимизируемым параметром при адаптации является погрешность (доли процента), а быстродействие является ограничением. В зависимости от этого выбираются параметры алгоритмов, такие как шаг интегрирования, размер сетки, критерии сходимости.
- параллельные вычисления. Многие численные методы допускают распараллеливание для работы на многоядерных процессорах или GPU. Это отдельная модификация алгоритма, позволяющая сократить или минимизировать обмен данными. Например, для группы сеточных методов минимизировать обмен между вычислительными узлами.
- работа с памятью. Реальные ИТ-задачи часто требуют критично больших объемов оперативной памяти. Адаптация в этом случае может включать использование методов, которые работают с данными во внешней памяти (out-of-core). Подобные техники сознательно переносят часть работы с данными на медленные устройства (например, HDD). Для этого исходная задача разбивается на отдельные блоки, блоки последовательно загружаются в оперативную память и выгружаются обратно на диск. Также используется оптимизация доступа к диску. Эти меры также сокращают время работы алгоритмов.

Данный вид адаптации актуален, например, для задач цифровой обработки сигналов, машинного обучения, линейной алгебры. Известна блочная реализация методов решения систем линейных уравнений (метод Гаусса, LU-разложение), работа систем управления базами данных (СУБД), обработка видео и др.

В. Верификация и валидация численного метода.

Суть этого этапа состоит как в проверке самой модели, полученной на этапе формализации задачи, так и в оценке правильности решения.

Верификация оценивает правильность реализации математической модели и численного метода. Для этого производят тестирование метода на задачах с имеющимся аналитическим решением. Сопоставление результатов производится при изменении параметров метода. Например, при изменении шага сетки, порядка метода и др.

Валидация – это процесс оценки точности и адекватности самой математической модели. Проведение численного анализа устанавливает соответствие результатов вычислений данным реальной задачи. Таким образом, суть этапа верификации и валидации заключается в установлении доверия к результатам численного моделирования. В результате формальная математическая модель сопоставляется с реальной физической задачей.

ИТ-реализация (программная инженерия и внедрение)

Цель этого этапа – подготовить задачу прикладной математики (представленной в виде алгоритма) к программной реализации, создать на основе алгоритма программный продукт или сервис для

решения исходной ИТ-задачи и обеспечить его готовность к промышленной эксплуатации [10].

На этапе ИТ-реализации производится материальная реализация математической модели и алгоритма в практический инструмент в виде программной системы. Как и при разработке любой сложной системы, этот процесс включает два ключевых этапа: проектирование и реализацию.

1. Проектирование программного приложения.

При проектировании программного приложения определяется архитектура будущей системы [11, 12]. Она определяет состав компонентов, их интерфейсы, взаимодействие между ними, а также среду функционирования и принципы разработки. Это сложный процесс, определяющий основные характеристики программной системы, и он включает:

А. Выбор стека технологий

- выбор языков программирования, который зависит от специфики задачи. Существуют устоявшиеся рекомендации по выбору языков: для научных вычислений и анализа данных часто выбирают Python; для высокопроизводительных задач, требующих максимальной эффективности, – C++, Go или Rust; для крупных корпоративных (enterprise) решений – Java или Kotlin;
- для специализированных вычислительных задач применяются специализированные библиотеки (например, TensorFlow или PyTorch для машинного обучения, Spring – для бизнес-приложений);
- выбор СУБД: реляционных (PostgreSQL, MySQL) или NoSQL (MongoDB, Redis) в зависимости от структуры и объема данных;
- выбор платформ для развертывания и интеграции, включая контейнеризацию, таких как Docker, Kubernetes, а также облачных сервисов (Yandex Cloud, AWS, Google Cloud Platform).

Б. Выбор типа архитектуры

Тип архитектуры или архитектурный стиль определяет высокоуровневую структуру приложения, организацию его компонентов и принципы их взаимодействия. Существуют разные классификационные признаки архитектуры. Например, по степени централизации могут быть монолитные и микросервисные (распределенные) системы. К числу современных и широко применяемых подходов относятся следующие типы архитектур:

- многослойная архитектура программной системы основана на принципе разделения ответственности путем вертикальной компоновки. Программное обеспечение состоит из набора слоёв (как правило, представления, бизнес-логики и доступа к данным), каждый из которых имеет строго определённую зону ответственности. Несмотря на внутреннюю модульность, система обычно реализуется в виде единого развертываемого модуля (монолита). Для организации взаимодействия внутри слоя представления часто применяются паттерны, такие как Model-View-Controller (MVC);
- многоуровневая архитектура системы делит программное обеспечение на уровни по принципу взаимоотношения «клиент–сервис». Уровни разграничивают ответственность поставщика данных и конечного потребителя. В зависимости от сложности задачи, системы могут быть одно-, двух-, трех- или многоуровневыми. Например, в классической двухуровневой клиент-серверной архитектуре клиентская часть объединяет логику представления и бизнес-логику, а серверная отвечает исключительно за хранение и управление данными;
- сервис-ориентированной архитектура (Service-Oriented Architecture – SOA) позволяет декомпозировать систему на независимые, слабосвязанные модули (сервисы). Взаимодействие между этими модулями устанавливается с помощью строго определенных сервисов. Этот подход обеспечивает высокую степень модульности, облегчает интеграцию разнородных систем и повышает гибкость всей ИТ-инфраструктуры предприятия;
- микросервисная архитектура развивает идеи SOA, представляя приложение как набор небольших независимых и автономных сервисов. Каждый микросервис ориентирован на выполнение одной бизнес-функции, работает в собственном процессе и может развертываться независимо от остальных. Взаимодействие между ними осуществляется по стандартному протоколу и формату данных, что обеспечивает высокую масштабируемость, отказоустойчивость и ускоряет циклы разработки.

Таким образом, грамотно проведенное проектирование позволяет создать надёжную и эффективно работающую программную систему, способную успешно справляться с поставленными задачами и адаптироваться к будущим изменениям требований бизнеса.

1. Реализация программного приложения.

На этапе реализации программной системы выбранные инструменты и подходы воплощаются в конкретном программном коде, тестируются и внедряются.

- Разработка программного кода

Разработчики пишут код, реализуя структуру и функциональные компоненты, необходимые файлы и классы, заложенные в ходе проектирования. Отслеживается соблюдение стандартов программирования, обеспечивается поддержка чистоты кода, соответствие стилю и принятым стандартам компании. В том числе с соблюдением принципов чистого кода и SOLID, обеспечивающих понятный, поддерживаемый и масштабируемый код.

- Тестирование

Разрабатываются тестовые сценарии для охвата максимального количества возможных вариантов поведения программы. На основе сценария выполняется комплекс автоматизированных и ручных тестов программного обеспечения. Юнит-тесты и интеграционные тесты направлены на проверку правильности выполнения функциональных требований. В ручном режиме проверяется интерфейс системы, удобство навигации и эргономичность интерфейсов.

- Интеграция и развертывание

Создается итоговый исполняемый файл, или контейнер (например, Docker), содержащий готовое решение. Производится интеграция всех компонентов и подготовка к последующему развертыванию в промышленной среде. Организация процессов непрерывной интеграции и доставки (полностью на англ – CI/CD) позволяет быстро вносить изменения и своевременно обновлять продукт.

- Документирование и сопровождение

После завершения разработки составляется подробная техническая документация, создаются инструкции пользователя и администрирования. Далее начинается фаза сопровождения по поддержке продукта, проводятся регулярные обновления, исправляются ошибки и добавляются новые возможности на обратной связи пользователей.

Таким образом, успешная реализация программной системы достигается сочетанием качественного проектирования и применения современных инженерных практик на этапе разработки.

Заключение

Изложенная концепция формализации ИТ-задач, результатом которой является промышленное программное решение, основана на системном подходе. Модель четырехэтапная, обеспечивает методологическую основу для формализации и включает последовательно: формализацию требований, математическое моделирование, выбор метода и алгоритмизацию и непосредственно ИТ-реализацию.

Ключевым моментом предлагаемого подхода является создание сквозного жизненного цикла, где каждый этап логически вытекает из предыдущего. Такой принцип обеспечивает целостность процесса, сохраняя логическую связь между первоначальной постановкой задачи и ее конечной программной реализацией.

СПИСОК ЛИТЕРАТУРЫ

1. Olivé A. Conceptual modeling of information systems. – Berlin, Heidelberg: Springer, 2007. – 455 p. DOI: doi.org/10.1007/978-3-540-39390-0.
2. Jackson M.A. Problem frames: analysing and structuring software development problems. – USA: Addison-Wesley, 2001. – 390 p.
3. Dubois C., Prevosto V., Burel G. Teaching formal methods to future engineers // Formal Methods Teaching. FMTea. Lecture Notes in Computer Science / Eds. B. Dongol, L. Petre, G. Smith. – Cham: Springer, 2019. – Vol. 11758. DOI: doi.org/10.1007/978-3-030-32441-4_5.
4. Моисеев Н.Н. Математические задачи системного анализа. – М.: ЛЕНАНД, 2023. – 609 с.
5. Благовар А. Обзор математики для начинающего ML-инженера // Habr. – 21 сентября 2025. URL: https://habr.com/ru/articles/942114/ (дата обращения: 15.12.2025).
6. Пестриков В.М. Математические методы в инженерии. – СПб.: ВШТЭ СПбГУПТД, 2023. – 158 с.
7. Мышкис А.Д. Прикладная математика для инженеров. Специальные курсы. 3-е изд. – М.: ФИЗМАТЛИТ, 2016. – 688 с.
8. Современные методы оптимизации и особенности их применения / С.М. Бекетов, Д.А. Зубкова, А.М. Гинцяк, Ж.В. Бурлуцкая, С.Г. Редько // Russian Technological Journal. – 2025. – Т. 13. – № 4. – С. 78–94. DOI: doi.org/10.32362/2500-316X-2025-13-4-78-94.
9. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ. 2-е изд. – М.: ИД «Вильямс», 2011. – 1296 с.
10. Окулов С.М. Программирование в алгоритмах. – М.: БИНОМ. Лаборатория знаний, 2002. – 341 с.
11. Ousterhout J. A Philosophy of Software Design. – Palo Alto, CA: Yaknyam Press, 2021. – 188 p.
12. Kleppmann M. Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems. – Sebastopol: O'Reilly Media, 2017. – 614 p.

Информация об авторах

Елена Алексеевна Кочегурова, кандидат технических наук, доцент отделения информационных технологий Инженерной школы информационных технологий и робототехники, Национальный исследовательский Томский политехнический университет, Россия, 634050, г. Томск, пр. Ленина, 30; kocheg@tpu.ru; https://orcid.org/0000-0003-4473-528X

Оксана Николаевна Ефремова, кандидат педагогических наук, доцент отделения математики и математической физики Инженерной школы ядерных технологий, Национальный исследовательский Томский политехнический университет, Россия, 634050, г. Томск, пр. Ленина, 30; oksananik@tpu.ru; https://orcid.org/0000-0001-5080-3280

Ольга Николаевна Имас, кандидат физико-математических наук, доцент отделения математики и математической физики Инженерной школы ядерных технологий, Национальный исследовательский Томский политехнический университет, Россия, 634050, г. Томск, пр. Ленина, 30; onm@tpu.ru; https://orcid.org/0000-0002-6068-0939

Анна Игоревна Шерстнёва, кандидат физико-математических наук, доцент отделения математики и математической физики Инженерной школы ядерных технологий, Национальный исследовательский Томский политехнический университет, Россия, 634050, г. Томск, пр. Ленина, 30; sherstneva@tpu.ru; https://orcid.org/0000-0002-6068-0939
Сергей Геннадьевич Цапко, кандидат технических наук, доцент отделения информационных технологий Инженерной школы информационных технологий и робототехники, Национальный исследовательский Томский политехнический университет, Россия, 634050, г. Томск, пр. Ленина, 30; tsapko@tpu.ru; https://orcid.org/0000-0003-2480-3847

Поступила в редакцию: 09.09.2025
Поступила после рецензирования: 20.11.2025
Принята к публикации: 27.12.2025

REFERENCES

1. Olivé A. *Conceptual modeling of information systems*. Berlin, Heidelberg, Springer, 2007. 455 p. DOI: doi.org/10.1007/978-3-540-39390-0.
2. Jackson M.A. *Problem frames: analysing and structuring software development problems*. USA, Addison-Wesley, 2001. 390 p.
3. Dubois C., Prevosto V., Burel G. Teaching formal methods to future engineers. *Formal Methods Teaching. FMTea. Lecture Notes in Computer Science*. Eds. B. Dongol, L. Petre, G. Smith. Cham, Springer, 2019. Vol 11758. DOI: doi.org/10.1007/978-3-030-32441-4_5.
4. Moiseev N.N. *Mathematical problems of systems analysis*. Moscow, LENAND Publ., 2023. 609 p. (In Russ.)
5. Blagovar A. A review of mathematics for a novice ML engineer. *Habr*, 2025, September 21. Available at: https://habr.com/ru/articles/942114/ (accessed 15 December 2025).
6. Pestrikov V.M. *Mathematical methods in engineering*. St Petersburg, VShTE SPbGUPTD Publ., 2023. 158 p.
7. Myshkis A.D. *Applied mathematics for engineers. Special courses*. 3rd ed. Moscow, FIZMATLIT Publ., 2016. 688 p.
8. Beketov S.M., Zubkova D.A., Gintsyak A.M., Burlutskaya Zh.V., Redko S.G. Modern optimization methods and features of their application. *Russian Technological Journal*, 2025, vol. 13, no. 4, pp. 78–94. DOI: doi.org/10.32362/2500-316X-2025-13-4-78-94.
9. Cormen T., Leiserson C., Rivest R., Stein K. *Algorithms: Construction and Analysis*. 2nd ed. Moscow, Williams Publ. House, 2011. 1296 p. (In Russ.)
10. Okulov S.M. *Programming in algorithms*. Moscow, BINOM. Laboratory of knowledge, 2002. 341 p. (In Russ.)
11. Ousterhout J. *A Philosophy of software design*. Palo Alto, CA, Yaknyam Press, 2021. 188 p.
12. Kleppmann M. *Designing data-intensive applications: the big ideas behind reliable*, *scalable*, *and maintainable systems*. Sebastopol, O'Reilly Media, 2017. 614 p.

Information about the authors

Elena A. Kochegurova, Cand. Sc., Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; kocheg@tpu.ru; https://orcid.org/0000-0003-4473-528X.
Oksana N. Efremova, Cand. Sc., Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; oksananik@tpu.ru; https://orcid.org/0000-0001-5080-3280
Olga, N. Imas, Cand. Sc., Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; onm@tpu.ru; https://orcid.org/0000-0002-6068-0939
Anna I. Sherstneva, Cand. Sc., Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; sherstneva@tpu.ru; https://orcid.org/0000-0002-6068-0939
Sergey G. Tsapko, Cand. Sc., Associate Professor, National Research Tomsk Polytechnic University, 30, Lenin avenue, Tomsk, 634050, Russian Federation; tsapko@tpu.ru; https://orcid.org/0000-0003-2480-3847.

Received: 09.09.2025
Revised: 20.11.2025
Accepted: 27.12.2025