Прахова Арина Евгеньевна,
студент кафедры
«Информационные системы и технологии»
ФГБОУ ВО ПГУТИ

МЕТОДОЛОГИЯ ФОРМАЛИЗАЦИИ НЕЧЕТКИХ ИТ-ТРЕБОВАНИЙ: ОТ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ К ВЕРИФИЦИРУЕМОЙ РЕАЛИЗАЦИИ

Аннотация. Статья посвящена разработке методологического каркаса поэтапной формализации ИТ-задач. Предложена пятиступенчатая модель трансформации требований от естественного языка к контрактному программированию с использованием темпоральных логик (LTL/CTL).

Ключевые слова: Программная инженерия, Формализация требований, Трансформация моделей, Темпоральная логика (LTL/CTL), Контрактное программирование, Предметно-ориентированные языки (DSL), Автоматическая генерация кода.

При введении любой сложной информационной системы неизбежно запускается процесс столкновения двух различных концептуальных рамок. С одной стороны, выступает заинтересованное лицо, оперирующее категориями предметной области и интуитивно понятными процессами. Его описание задачи всегда контекстуально, неполно и зачастую содержит скрытые противоречия. С другой стороны находится вычислительная среда, требующая абсолютной точности, полноты описания всех состояний и предсказуемости реакций. Именно в пространстве между этими двумя полюсами появляется главная цель большинства программных продуктов. Актуальность глубокого переосмысления процесса формализации продиктована тем, что современные отказоустойчивые сервисы больше не могут полагаться на тестирование на выходе как единственный механизм обеспечения качества. Цена ошибки в логике высоконагруженных систем требует переноса доказательного аппарата на самые ранние этапы жизненного цикла разработки.

Обзор существующих подходов к решению данной проблемы демонстрирует эволюцию инженерной мысли от чисто текстовых методов к строгому математическому моделированию. Использование естественного языка, поддающегося контролю, позволяет лишь частично снять лексическую неоднозначность, но оказывается бессильным перед описанием параллельных процессов и свободного доступа к данным. Попытки регламентировать визуальные обозначения сталкиваются с размытой семантикой самих стандартов, где один и тот же элемент может интерпретироваться генератором кода десятком разных способов. Более зрелым направлением представляется применение сетей Петри для анализа достижимости состояний и темпоральных логик (LTL/CTL) для доказательства непредсказуемого поведения систем. Однако исторически эти методы оставались принципом для академических исследований из-за высокого порога входа для практикующих разработчиков. Современный компромисс предлагает концепция Model-Driven Engineering, предполагающая поднятие уровня детализации до метамоделей, а также развитие предметно-ориентированных языков (*DSL*), которые скрывают синтаксическую сложность формальных спецификаций за терминами конкретной индустрии.

Ниже представлена таблица многоуровневой модели формализации, состоящая из пяти дискретных этапов, служащих для систематизации процесса превращения идеи в код.

Таблица 1

Многоуровневая модель формализации ИТ-задач

Уровень	Название	Инструментарий / Нотация	Цель
L0	Неформальная постановка	Естественный язык, интервью, майнд-карты	Фиксация субъективного видения стейкхолдеров
L1	Концептуальная модель	Онтологии (OWL), ER-диаграммы, глоссарии	Устранение терминологической путаницы, определение сущностей и связей
L2	Формальная спецификация поведения	Темпоральные логики (LTL/CTL), сети Петри, TLA+	Математическое описание динамики системы без привязки к языку программирования
L3	Алгоритмическая модель	Абстрактные типы данных, псевдокод, контрактное программирование (Design by Contract)	Определение алгоритмов и интерфейсов взаимодействия компонентов
L4	Программная реализация	Исходный код (Rust/Go/TypeScript), DSL, автоматическая генерация через MDA/MDE	Исполняемый артефакт

Переход между данными уровнями не должен быть самопроизвольным. Каждый шаг представляет собой этап проектирования, сопровождаемый потерей части информации, которая намеренно отсекается как несущественная. Ключевым моментом всей методологии является трансформация второго уровня в третий. Свойства безопасности системы, выраженные формулами линейной темпоральной логики (например, утверждение о том, что система никогда не останется в тупике), должны транслироваться в предусловия и постусловия функций на уровне алгоритмической модели. Это реализует принцип контрактного программирования, где каждая единица кода несет в себе соответствующее математическое доказательство своей корректности в рамках заданных входных данных. Таким образом, разработчик перестает писать код вслепую и начинает реализовывать уже доказанную математическую модель.

Программная реализация формализованных моделей сегодня опирается на несколько механизмов, дополняющих друг друга. Автоматическая генерация исходного кода является самым простым путём, когда из статической спецификации извлекаются сигнатуры типов, структуры данных и пустые тела функций, из которых инженеру остаётся написать реальный программный код. Однако более чётким подходом представляется использование мощности современных систем типов. Языки со строгой статической типизацией и поддержкой зависимых типов позволяют перенести значительную часть доказательств в сам компилятор. Если программа успешно прошла проверку типов в системе вроде Idris или Haskell с расширениями, это само по себе служит гарантией того, что определенный класс логических ошибок в ней невозможен.

Особое место занимает отечественная школа программирования с ее методами суперкомпиляции и средой разработки на базе Рефала. Данные техники рассматривают программу не как набор инструкций для процессора, а как систему рекурсивных уравнений над символьными выражениями. Путем эквивалентных преобразований спецификация высокого уровня сворачивается в оптимальный исполнимый код, причем сам процесс преобразования может быть заранее доказан. Это направление идеально ложится в основание рассматриваемой концепции, так как полностью стирает границу между спецификацией и реализацией, делая их единым целым.

Предложенная иерархическая модель существенно снижает когнитивную нагрузку на инженера, что позволяет применять средства автоматического доказательства теорем на тех этапах, где процесс внесения изменений не становится критичным. Разделение уровней L1 и L2 решает вечный конфликт между бизнесом и разработкой: аналитик работает над полнотой концептуальной схемы домена, не отвлекаясь на вопросы многопоточности, в то время как архитектор проектирует протоколы взаимодействия, опираясь на готовую карту терминов. Перспективы дальнейшего развития методики лежат в области интеграции больших языковых моделей. Нейросети способны взять на себя черновую работу по первичному переводу расшифровок встреч уровня L0 в структурированные прототип концептуальных моделей уровня L1, после чего подключается человек для проверки смыслов и перевода их в строгую математическую модель уровня L2.

В заключение следует отметить, что формализация ИТ-задач перестает быть только теоретическим знанием и превращается в практичный инструмент управления техническим риском. Переход от интуитивного написания строк кода к конструированию проверяемых моделей обозначает новый этап роста программной инженерии. Будущее промышленной разработки лежит именно в этой сфере, где естественная человеческая неопределенность последовательно преобразовывается в строгие ограничения, пока не превратится в безупречно работающий алгоритм, каждый бит которого имеет под собой математическое основание. Дальнейшие исследования целесообразно направить на создание автоматизированных валидаторов, которые способны оценивать степень соответствия между уровнем бизнес-понятий и конечным кодом

Список литературы:

1. Камкин А. С. Введение в формальные методы верификации программ [Текст]. – М.: ДМК-Пресс, 2024. – 304 с.

2. Кларк Э.М., Грамберг О., Пелед Д. Верификация моделей программ. Model Checking [Текст]: пер. с англ. – М.: МЦНМО, 2002. – С. 26–32, 56–61, 64–79.

3. Lathouwers S., Zaytsev V. Modelling Program Verification Tools for Software Engineers [Текст]: электрон. ресурс // Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems. – Montreal, Quebec, Canada: Association for Computing Machinery, 2022. – С. 98–108. Режим доступа: https://dl.acm.org/doi/abs/10.1145/3550355.3552426