

Математические и статистические методы

ДИАЛЕКТИЧЕСКИЙ МЕТОД ГЕГЕЛЯ В АНАЛИЗЕ РАЗВИТИЯ ИНДУСТРИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Алиев Бейлак Намаз оглы
МГУ имени М.В. Ломоносова, экономический факультет
(г. Москва, Россия)

Аннотация

В статье представлен методологический подход к анализу эволюции индустрии разработки программного обеспечения на основе диалектического метода Г.В.Ф. Гегеля. Центральным аналитическим принципом выступает представление о развитии через внутреннее противоречие, отраженное в логике становления и снятия (Aufhebung) противоположностей как ключевом механизме саморазвития понятий и систем. В рамках исследования гегелевский подход соотносится с положениями, сформулированными впоследствии в марксистской философии как диалектические законы – развитие через противоречия (вместо противопоставления «единства и борьбы противоположностей»), переход количественных изменений в качественные и отрицание отрицания – интерпретируемые в данной работе как инструментальные модели описания этапов трансформации технологических и организационных парадигм в сфере разработки ПО. Особое внимание уделяется применению принципа движения от абстрактного к конкретному, в том числе в трактовке Э.В. Ильенкова, для формализации историко-логического анализа переходов между методологиями разработки (Waterfall, Agile, DevOps) и современных форм, обусловленных распространением генеративных нейросетей, low-code/no-code решений и интеллектуальной автоматизации. Обосновывается научная и практическая ценность применения диалектического метода в контексте анализа высокотехнологичных индустрий. Работа носит философско-методологический характер и ориентирована на формирование операциональной основы для дальнейших эмпирических и имитационных исследований. Также рассматриваются ограничения предлагаемого подхода и направления дальнейших исследований, включая потенциал прогнозирования будущих изменений в индустрии ПО под воздействием систем искусственного интеллекта.

Ключевые слова: диалектический метод, Гегель, разработка программного обеспечения, эволюция индустрии ПО, методологии разработки ПО, Waterfall, Agile, DevOps, информатика, LLM, AI, low-code, no-code.

JEL коды: C18, C63, C87.

Для цитирования: Алиев Б.Н. Диалектический метод Гегеля в анализе развития индустрии разработки программного обеспечения // Научные исследования экономического факультета.

Введение

Индустрия разработки программного обеспечения (ПО) представляет собой одну из наиболее динамично развивающихся отраслей современной экономики (по оценкам, мировой рынок ПО составил ~730,7 млрд долл. в 2024 г. с прогнозом роста CAGR 11,3 %) (Grand View Research, 2025), оказывая фундаментальное влияние на все сферы человеческой деятельности. Тенденции цифровизации, автоматизации, внедрения IoT и индустриального ПО стимулируют дальнейший рост: например, рынок промышленного программного обеспечения в 2023 г. оценивался в 146 млрд долл., с прогнозом к 355 млрд долл. к 2030 г. (CAGR $\approx$ 13,5 %) (IoT Analytics, 2025). Стремительное технологическое обновление, появление новых инструментов и методологий разработки, а также смена парадигм требуют глубокого философско-методологического осмысления закономерностей происходящих трансформаций. В существующих подходах к анализу эволюции индустрии ПО акцент, как правило, делается на технологических, инженерных или экономических аспектах, в то время как философские основания развития часто остаются вне поля аналитического рассмотрения.

Актуальность настоящего исследования обусловлена необходимостью выработки междисциплинарной методологии, способной не только описывать, но и объяснять, а также прогнозировать траектории развития индустрии ПО как сложной саморазвивающейся системы. В качестве методологического основания предлагается использование диалектического метода, разработанного Г.В.Ф. Гегелем, центральным элементом которого выступает представление о развитии через внутренние противоречия как источнике движения и саморазвития понятий и форм. Гегелевская диалектика трактует развитие не как линейный процесс, а как развертывание логических противоположностей, чье снятие (*Aufhebung*) приводит к возникновению новых, более сложных форм.

Понимание процессов развития через внутренние противоречия, лежащие в основе гегелевского метода, находит развитие в марксистской философии, в рамках которой были сформулированы так называемые три закона диалектики: закон единства и борьбы противоположностей, закон перехода количественных изменений в качественные, а также закон отрицания отрицания (Энгельс, 1961). В настоящем исследовании они рассматриваются не как прямое выражение гегелевской логики, а как интерпретации и формализации, полезные для анализа реальных исторических и технологических процессов. Такой подход позволяет расширить границы применения диалектического метода, интегрируя его с современными инструментальными средствами анализа.

Проблема исследования заключается в отсутствии устойчивых методологических схем, объединяющих философскую диалектику и количественные методы, применимые к анализу эволюции высокотехнологичных индустрий. Возникает необходимость в адаптации диалектических категорий и принципов (в частности, принципа перехода от абстрактного к конкретному, разработанного Э.В. Ильенковым в русле гегелевско-марксистской традиции) для целей экономико-математического моделирования.

Цель исследования – разработка и теоретическое обоснование методологического под-

хода к анализу эволюции индустрии разработки программного обеспечения на основе диалектического метода Гегеля и его применимости к формализации процессов технологического развития, а также определение путей его формализации для использования в рамках математических, статистических и инструментальных методов в экономике.

Научная новизна исследования заключается в следующем.

1. Предлагается комплексный междисциплинарный подход к анализу развития индустрии ПО, сочетающий гегелевскую диалектику, принципы логического становления и снятия противоречий, концепцию перехода от абстрактного к конкретному (Ильенков), а также современные методы экономико-математического моделирования.

2. Обосновывается возможность применения диалектических закономерностей (в форме, разработанной в марксистской философии) для анализа исторической смены методологических и организационных моделей в индустрии ПО (например, переход от Waterfall к Agile и далее к DevOps).

Намечаются пути формализации диалектических принципов в виде моделей и аналитических схем, пригодных для описания и прогнозирования траекторий развития индустрии программной инженерии с использованием количественных методов.

1. Обзор литературы

Анализ существующих публикаций показывает, что большинство исследований сосредоточены на технологических, управленческих и эмпирических аспектах ПО. В то же время философские и методологические подходы в ИТ достаточно слабо представлены – об этом свидетельствует обзор систем исследований ИТ-парадигм и обзор методических парадигм в ИТ (Systematic Review) (Kirongo, 2020; Swaleh Mohammed, 2025). Работы, применяющие философскую методологию, в частности диалектику, для анализа развития индустрии ПО, единичны и носят преимущественно описательный характер. Исследования, связывающие диалектический метод с математическим моделированием экономических процессов в ИТ-сфере, практически отсутствуют, что подчеркивает актуальность и новизну данной работы. В философии технологий также отмечается разнообразие методологических подходов, и многие работы подчеркивают важность такого разнообразия (Joseph, 1996). Необходимо проведение глубокого и критического обзора литературы для выявления существующих подходов к моделированию технологических сдвигов и оценки возможности интеграции диалектических принципов в эти модели. Например, исследования по диффузии инноваций или модели технологических жизненных циклов могут быть обогащены диалектической перспективой, другими словами, традиционные исследования о том, как распространяются технологии или как они развиваются со временем, можно сделать глубже, если рассматривать не только линейные процессы, но и внутренние противоречия, конфликты и изменения, которые происходят в этих процессах. Работа по философии информационной цивилизации и информационных технологий (Абдеев, 1994) рассматривает диалектику в контексте развития информационного общества и технологий. Хотя это не прямой анализ индустрии ПО через диалектический метод Гегеля, она закладывает философские основы для такого анализа. Публикации, в которых диалектический подход используется применительно к ИТ, немногочисленны и достаточно разнородны. Часть из них связана не с программной инженерией как таковой, а с более общими вопросами цифровой и информационной среды. Так, в работе (Емельяненко, Яненко, 2021) диалектика рассматривается в контексте познания информации, получаемой из интернета. Для

настоящего исследования эта работа важна скорее как философско-методологический ориентир, чем как прямой анализ индустрии разработки ПО. Иной характер имеют публикации (Wernick et al., 2000; Robinson, 1998), где диалектическая проблематика уже соприкасается с вопросами разработки и моделирования информационных систем. Однако и здесь речь идет не о целостном анализе эволюции индустрии ПО, а лишь об отдельных аспектах методологии. Наконец, в работах (Мыльник, 2009; Беркова, 2022) диалектика используется в более широком контексте технологического и экономического развития. Такие исследования не дают прямой модели для анализа программной индустрии, но позволяют уточнить методологические основания, на которые может опираться настоящее исследование.

Важно отметить, что многие из этих работ:

- 1) описательны: они используют диалектику как способ интерпретации или описания определенных явлений, а не как строгий метод для построения проверяемых моделей;
- 2) фрагментарны: фокусируются на отдельных аспектах (например, устаревшее ПО, конкретная методология, процесс моделирования), а не на всей индустрии;
- 3) метафоричны: иногда термин «диалектика» используется для обозначения любого процесса, включающего противоречия и развитие, без глубокой привязки к гегельянской системе.

2. Методы

Данное исследование базируется на диалектическом методе, разработанном Г.В.Ф. Гегелем, в центре которого находится развитие через внутренние противоречия и их снятие (*Aufhebung*). Мы выбираем этот метод анализа, поскольку он позволяет связать философскую категориальную рефлексию с экономико-математическим моделированием. Существует множество методологических подходов в философии технологий (Joseph, 1996), но именно диалектический метод наиболее адекватен для отображения динамики, противоречивости и переходов между уровнями абстракции и конкретности. Гегель понимал развитие не как линейный процесс, а как последовательное разворачивание и преодоление противоположностей, порождающее новые, более сложные формы.

Для целей анализа индустрии разработки ПО в работе также привлекаются положения марксистской философии (Энгельс), где гегелевская логика была интерпретирована и диалектически реконструирована в виде трех «законов диалектики». В контексте данной статьи они рассматриваются не как собственные законы Гегеля, а как инструментальные схемы описания, полезные для интерпретации реальных историко-технологических процессов (Гегель, 1970):

1. *Развитие через противоречия (традиционно формулируется как «единство и борьба противоположностей»)*: конфликты между стабильностью и инновациями, скоростью и качеством, потребностями пользователей и техническими ограничениями.
2. *Переход количественных изменений в качественные*: накопление изменений (рост сложности проектов, объема данных, размера команд) ведет к скачкообразным трансформациям (смена архитектур, внедрение новых методологий).
3. *Отрицание отрицания*: каждая новая стадия снимает предшествующую – отрицает ее односторонность, но сохраняет рациональные элементы, выводя систему на новый уровень (например, переход от Waterfall к Agile и далее к DevOps).

При этом следует учитывать, что схема «тезис – антитезис – синтез» не используется в работе как буквальная модель гегелевской логики. В «Науке логики» развитие понятий осуществляется не через внешнюю последовательность стадий, а через внутреннее самоопределение категорий и их снятие. В настоящем исследовании триадическая схема применяется исключительно как эвристическое средство структурирования материала, тогда как методологическое основание анализа составляет идея имманентного движения организационной формы.

Необходимо особо подчеркнуть, что в настоящем исследовании противоречие не отождествляется с эмпирическим конфликтом интересов или показателей. Конфликты между скоростью и качеством, разработкой и эксплуатацией, инновацией и устойчивостью рассматриваются как феноменальные проявления более глубокой структурной несовместимости принципов организации разработки. В гегелевском смысле противоречие понимается как имманентный момент самой организационной формы: реализуя собственные основания, она одновременно порождает ограничения, которые в определенных условиях делают дальнейшее развитие в прежних рамках невозможным. Эмпирические конфликты фиксируют лишь внешнее выражение этой внутренней определенности.

Следует также разграничить логический и историко-описательный уровни анализа. В рамках настоящей работы диалектика используется не как схема ретроспективного упорядочивания эмпирических стадий, а как способ реконструкции внутренней необходимости переходов между организационными формами. Историческая последовательность методологий и архитектур рассматривается постольку, поскольку в ней проявляется движение понятий – изменение способов координации, управления сложностью и распределения ответственности. Тем самым анализ направлен не на описание событий как таковых, а на выявление логики их становления.

Таким образом, методология исследования сочетает гегелевскую логику становления и снятия с инструментальными моделями, разработанными в марксистской традиции, что позволяет интегрировать философский анализ с количественными методами. Применение диалектического метода к анализу индустрии ПО предполагает выявление ключевых противоречий на каждом этапе ее развития и анализ механизмов их разрешения, которые приводят к качественным скачкам и появлению новых форм.

3. Формализация диалектического подхода и инструментальные методы

Для придания диалектическому анализу операционального характера и возможности его использования предлагается опираться на принцип восхождения от абстрактного к конкретному, сформулированный Э.В. Ильенковым (Ильенков, 1960), который предполагает движение от общих категорий к целостной системе понятий, отражающих внутренние связи предмета. Этот принцип предполагает движение от общих философских категорий к конкретным, измеряемым параметрам и моделям.

Предлагаются следующие направления для математической формализации и моделирования.

1. Динамическая модель эволюции методологий разработки ПО:

1.1. Переменные. В качестве ключевых переменных модели рассматриваются стоимость разработки $C(t)$, средняя длительность проекта $T(t)$, качество или стабильность системы

$S(t)$ (например, число дефектов на единицу кода), скорость поставки ценности $V(t)$ (в частности, частота релизов) и уровень автоматизации $R(t)$.

1.2. Модель. Для описания взаимосвязи указанных переменных используется система дифференциальных или разностных уравнений (формула 1). В модель также вводится показатель «инновационного конфликта», или диалектического противоречия, $\Delta(t)$, который характеризует степень рассогласования между требованиями к скорости разработки и фактическим состоянием системы по качеству. В простейшем случае этот показатель может быть задан как функция разрыва между требуемой скоростью поставки $V(t)$ и текущим уровнем качества $S(t)$ (формула (1)). Когда $\Delta(t)$ достигает критического значения, система утрачивает устойчивость, что в рамках модели трактуется как качественный переход – смена доминирующей методологии разработки (формула 2).

1.3. Пример алгоритма (концептуальная схема):

1.3.1. На начальном этапе системе задаются параметры, соответствующие доминирующей методологии разработки. Так, для Waterfall могут быть характерны высокое значение $T(t)$, умеренный уровень $C(t)$ и сравнительно низкая гибкость процесса.

1.3.2. Затем по мере роста требований к скорости и адаптивности в системе накапливаются количественные изменения (например, рост требований к скорости и гибкости).

1.3.3. Далее, происходит увеличение «противоречия» $\Delta(t)$ между текущими возможностями методологии и новыми требованиями.

1.3.4. При $\Delta(t) > \Delta_{crit}$, происходит «качественный скачок» – переход к новой методологии (например, Agile), характеризующейся новым набором параметров (например, снижение $T(t)$, рост $V(t)$, возможное временное снижение $S(t)$ до стабилизации) (формула (3)).

Воспроизводимость исследования может быть обеспечена путем запуска кода на Python приведенного в приложении 1.

Рассмотрим переменные системы:

Пусть на момент времени t значения ключевых параметров характеризуют состояние методологии разработки ПО:

$C(t)$ – стоимость разработки,
 $T(t)$ – средняя длительность проекта,
 $S(t) = \frac{\text{качество}}{\text{стабильность}}$ (например, количество дефектов на $KLOC$),
 $KLOC$ – тысячи строк кода,
 $V(t)$ – скорость поставки ценности (например, релизов в единицу времени),
 $R(t)$ – уровень автоматизации процессов,

$$\left\{ \begin{array}{l} \frac{dC}{dt} = \alpha_1 \cdot V(t) - \alpha_2 \cdot R(t) - \alpha_3 \cdot S(t) \\ \frac{dT}{dt} = -\beta_1 \cdot R(t) + \beta_2 \cdot S(t) \\ \frac{dS}{dt} = -\gamma^1 \cdot V(t) + \gamma^2 \cdot R(t) \\ \frac{dV}{dt} = \delta^1 \cdot R(t) - \delta^2 \cdot S(t) \\ \frac{dR}{dt} = f(C(t), V(t), S(t), T(t)), \end{array} \right. \quad (1)$$

где $\alpha_i, \beta_i, \gamma_i, \delta_i \in R_+$ – параметры модели, определяемые эмпирически или экспертно.

Показатель противоречия (диалектического напряжения)

$$\Delta(t) = w_1 \cdot ((V(t)/V_{\text{цель}}) - 1)^2 + w_2 \cdot ((S_{\text{цель}}/S(t)) - 1)^2, \quad (2)$$

где:

$V_{\text{цель}}$ – желаемая (требуемая) скорость поставки,

$S_{\text{цель}}$ – целевое качество,

w_1, w_2 – веса значимости факторов.

Механизм фазового перехода:

$$\Delta(t) > \Delta_{\text{крит}} \Rightarrow \text{фазовый переход: смена методологии} \quad (3)$$

или

$$(C, T, S, V, R)|_{t+} = (C', T', S', V', R').$$

Для иллюстрации логики модели был проведен демонстрационный вычислительный эксперимент. Его задача состояла не в точном прогнозировании, а в наглядном воспроизведении механизма накопления внутренних противоречий в системе разработки программного обеспечения. Полученные временные ряды позволяют показать предполагаемый характер изменения ключевых переменных. На рисунке 1 представлена динамика скорости поставки, на рис. 2 – динамика качества, на рис. 3 – накопление диалектического противоречия.



Рисунок 1. Динамика скорости поставки (составлено автором)

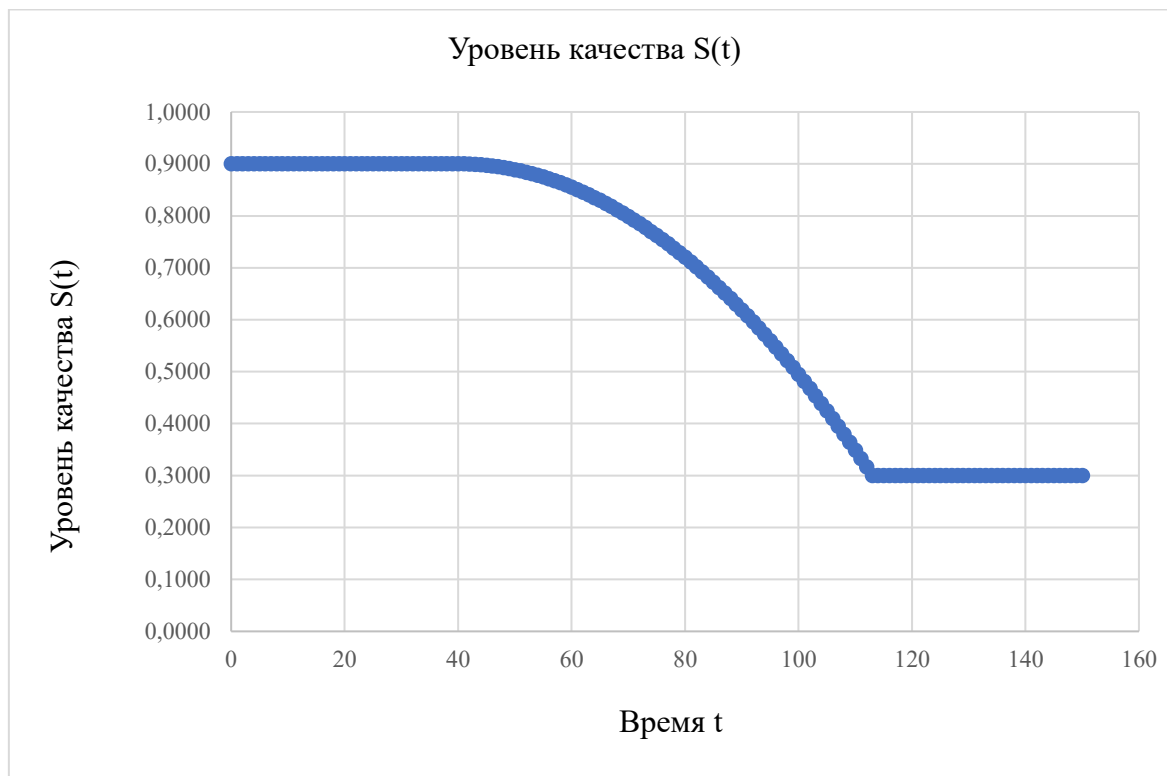


Рисунок 2. Динамика уровня качества (составлено автором)



Рисунок 3. Накопление диалектического противоречия (составлено автором)

Следует подчеркнуть, что приведенные зависимости имеют иллюстративный характер. Они отражают гипотезу о том, что координационные и организационные напряжения сначала нарастают постепенно, а затем, при достижении критического порога, приводят к качественному скачку. Поэтому в данной версии работы форма кривых, наклон линий и масштабы осей не привязаны к эмпирическим данным и используются прежде всего для визуализации внутренней логики процесса.

Динамика скорости поставки, представленная на рис. 1, показывает устойчивый рост по мере увеличения уровня автоматизации. Такой рост отражает тенденцию к ускорению процессов за счет автоматизации, CI/CD и практик DevOps, при этом рост не является бесконечным, так как по мере приближения к верхним значениям проявляется эффект насыщения, связанный с внутренними ограничениями системы. Это позволяет предположить, что на ранних этапах количественное ускорение маскирует накапливающиеся противоречия, которые позднее становятся существенными.

Иная картина наблюдается в динамике качества, представленной на рис. 2, на начальном этапе качество остается относительно стабильным, несмотря на рост скорости поставки, однако после прохождения определенного порога дальнейшее ускорение начинает сопровождаться его постепенным снижением. В содержательном плане это можно трактовать как проявление внутреннего противоречия между стремлением к ускорению разработки и необходимостью сохранять надежность программного продукта. В практике разработки подобная ситуация обычно связана с накоплением технического долга и усложнением сопровождения системы.

На рисунке 3 показана динамика индекса диалектического противоречия $\Delta(t)$, который агрегирует отклонения системы от условно нормативных значений скорости и качества. Именно этот показатель наиболее отчетливо отражает момент, когда накопленные количественные изменения начинают перерастать в качественную трансформацию системы.

Динамика показателя $\Delta(t)$ демонстрирует нелинейный характер накопления противоречий. На ранних этапах развитие системы сопровождается сравнительно низким уровнем внутреннего напряжения, однако по мере дальнейшего количественного роста скорость накопления противоречий возрастает. При достижении критического порога Δ_{crit} система входит в состояние неустойчивости, которое в рамках модели интерпретируется как момент качественного перехода. Тем самым предложенная схема показывает, что диалектический анализ может быть операционализирован в форме вычислительного эксперимента. Такой эксперимент не заменяет эмпирическую проверку, но позволяет формализовать и наглядно представить философские категории развития, противоречия и качественного скачка. В этом смысле диалектический метод Гегеля может рассматриваться как метатеоретический инструмент анализа эволюции процессов разработки программного обеспечения, а предложенная метамодель – как основа для дальнейших исследований, связанных с построением более детализированных имитационных моделей.

2. Модель «развития через противоречие» на основе моделирования агентов.

2.1. Можно представить различных стейкхолдеров (разработчики, менеджеры, пользователи) как агентов с различными, порой противоречивыми, целями. Разрешение противоречий может моделироваться как поиск равновесия в динамической игре или как результат адаптивного поведения агентов. В будущих исследованиях рекомендуется комбинировать агентское моделирование с RL (обучение с подкреплением) или генетическими алгоритмами, такой подход позволит эволюционировать модели, что позволит избежать обучения модели под ожидания. Такой подход позволит приблизить моделирование к реальной (приближенной к реальному) системе (Алиев, 2024).

Рассмотрим источники и качество данных, критерии отбора.

Для эмпирической верификации и калибровки моделей предполагается использование следующих источников данных:

- 1) ежегодные отчеты и опросы разработчиков: JetBrains Developer Ecosystem Survey, Stack Overflow Developer Survey, GitHub State of the Octoverse;
- 2) аналитические отчеты: ThoughtWorks Technology Radar, отчеты Gartner, Forrester;
- 3) данные репозиторий ПО (например, GitHub) для анализа трендов использования технологий, языков программирования, активности проектов;
- 4) публикации в научных базах (Scopus, WoS, BAK) для анализа трендов в исследованиях методологий и инструментов.

Критерии отбора данных:

- 1) релевантность тематике исследования (методологии разработки, инструменты, экономические показатели ИТ-проектов);
- 2) временной охват, позволяющий отследить динамику (желательно за последние 10–15 лет);
- 3) достоверность и репрезентативность источников (приоритет отдается крупным международным опросам и авторитетным аналитическим агентствам).

Обратимся к концептуальной схеме эмпирической калибровки модели.

Для перевода предложенной метамодели из статуса демонстрационного вычислительного эксперимента в инструмент аналитического исследования требуется концептуальная операционализация ее переменных и параметров. В данном подразделе намечается возможная схема эмпирической калибровки модели на основе доступных индустриальных и репозиторных данных.

Для эмпирического наполнения модели переменную скорости поставки ценности $V(t)$ можно связать с метриками частоты деплоя (Deployment Frequency) и времени прохождения изменения от коммита до вывода в продакшен (Lead Time for Changes), которые широко используются в индустриальных исследованиях, в том числе в отчетах программы DevOps Research and Assessment (DORA). В качестве дополнительных прокси-показателей могут использоваться средняя частота релизов, интенсивность merge-операций в основной ветке репозитория и средняя продолжительность обработки пользовательских задач (cycle time). Переменная качества и устойчивости системы $S(t)$ может быть операционализирована через долю неуспешных изменений (Change Failure Rate), среднее время восстановления после инцидента (Mean Time to Recovery), плотность дефектов на тысячу строк кода, частоту откатов релизов, а также динамику повторно открытых задач. Совокупность этих показателей позволяет количественно оценивать уровень системной стабильности в условиях ускорения разработки.

Для оценки средней длительности проектного цикла $T(t)$ могут использоваться медианное время закрытия задач, длительность спринтов, средняя продолжительность жизненного цикла feature-ветки и агрегированные показатели time-to-market для отдельных релизов. Уровень автоматизации $R(t)$ целесообразно описывать через долю автоматизированных тестов, степень использования CI/CD-пайплайнов, распространенность практик Infrastructure as Code, а также через интегральные показатели внедрения DevOps-инструментов и процессов.

Отдельного пояснения требует индекс диалектического противоречия $\Delta(t)$, формально заданный в разделе 3 (формула (2)). В рамках эмпирической калибровки его можно рассматривать как агрегированный показатель рассогласования между целевыми и фактическими значениями ключевых параметров системы разработки.

Целевые значения $V(target)$ и $S(target)$ могут задаваться на основе индустриальных бенчмарков, например по медианным значениям верхнего квартиля высокоэффективных команд. Фактические значения $V(t)$ и $S(t)$, в свою очередь, могут извлекаться из репозиторных данных и внутренних метрик организации. Весовые коэффициенты, отражающие относительную значимость скорости и качества, целесообразно определять либо на основе регрессионного анализа, либо посредством экспертной калибровки. В таком виде ранее введенная формальная конструкция $\Delta(t)$ приобретает эмпирически интерпретируемое содержание и может использоваться как измеримый показатель системного напряжения, накапливающегося по мере количественных изменений в процессе разработки. При этом важно подчеркнуть, что предлагаемая формализация не претендует на исчерпывающее математическое выражение диалектической логики. Индекс противоречия и связанные с ним зависимости рассматриваются прежде всего как инструмент моделирования структурной динамики организационных форм, а не как попытка свести движение понятий к набору количественных индикаторов. Иначе говоря, формальная модель фиксирует лишь отдельные аспекты процесса – прежде всего накопление и разрешение системного напряжения, - но не подменяет собой философскую интерпретацию.

4. Результаты

Представим применение диалектического метода к истории развития индустрии ПО.

Эволюция индустрии ПО может быть представлена как последовательность диалектических циклов, где каждый этап порождает противоречия, разрешение которых ведет к новому качественному состоянию.

Следует подчеркнуть, что предложенная периодизация не претендует на новизну в эмпирическом смысле – выделяемые этапы в целом соответствуют общепринятой исторической картине развития отрасли. Однако в рамках настоящего исследования принципиально важным является не само перечисление стадий, а их интерпретация как логически необходимых форм развертывания внутреннего противоречия индустрии разработки ПО.

Исторические переходы рассматриваются не как случайная последовательность технологических инноваций, а как процесс имманентного саморазвития системы, в котором каждая последующая форма возникает из ограниченности предыдущей. В этом смысле методологический акцент переносится с описания внешних событий на реконструкцию внутренней логики становления – выявление того, какие именно противоречия делали прежнюю организационную форму неустойчивой и тем самым порождали необходимость ее снятия.

Такой подход позволяет трактовать развитие индустрии разработки ПО как движение от абстрактных и односторонних форм организации к более конкретным и дифференцированным структурам, способным удерживать возрастающую сложность цифровой экономики.

1. 1940–1950-е: рождение программирования (тезис)

1.1. *Состояние*: программирование на машинных кодах, тесная связь ПО и аппаратного обеспечения.

1.2. *Противоречие*: трудоемкость и низкий уровень абстракции машинных кодов противостоят растущей потребности в решении все более сложных задач.

1.3. *Разрешение (синтез, становящийся новым тезисом)*: появление первых языков высокого уровня (Фортран, Лисп, Кобол), абстрагирующих от аппаратной специфики (Lee J., 1981).

2. *1960–1970-е: программный кризис (антитезис к неструктурированному развитию)*

2.1. *Состояние*: рост сложности ПО, срыв сроков и превышение бюджетов («программный кризис» термин вводит (Randell, 1996) для описания возникших проблем со сложностью).

2.2. *Противоречие*: необходимость создания крупных и надежных программных систем против отсутствия формализованных методов управления разработкой и обеспечения качества.

2.3. *Разрешение*: зарождение инженерии ПО (Software Engineering), появление структурного программирования, концепций жизненного цикла ПО. IBM отделяет ПО от оборудования (1969), признавая ПО самостоятельным продуктом (Johnson, Luanne, 1998).

3. *1970–1990-е: ПО как индустрия и массовый рынок (синтез и новый тезис)*

3.1. *Состояние*: формирование рынка коммерческого ПО, появление персональных компьютеров, распространение проприетарного ПО и зарождение open source.

3.2. *Противоречие*: монолитность и сложность поддержки крупных приложений с одной стороны, а с другой потребность в быстрой адаптации и кастомизации под нужды массового пользователя; ограничения модели «водопада» (Waterfall) в условиях быстро меняющихся требований.

3.3. *Разрешение*: появление объектно-ориентированного программирования, развитие интегрированных сред разработки (IDE), зарождение итеративных подходов.

4. *1990–2000-е: веб-технологии и интернет (качественный скачок, антитезис к традиционному ПО)*

4.1. *Состояние*: бурное развитие интернета, появление веб-приложений, глобальное распространение ПО.

4.2. *Противоречие*: традиционные десктопные приложения и длительные циклы разработки против потребности в мгновенной доставке и обновлении ПО, интерактивности и доступности из любой точки.

4.3. *Разрешение*: развитие веб-технологий, рост open source (Apache, MySQL), формирование гибких методологий разработки (Agile).

5. *2010-е: облака, мобильные приложения, Agile и DevOps (синтез)*

5.1. *Состояние*: доминирование мобильных платформ, облачных вычислений (SaaS, PaaS, IaaS), широкое распространение Agile и становление DevOps (Almeida F, 2022).

5.2. *Противоречие*: ускорение циклов разработки (Agile) «слева», а «справа» традиционные барьеры между разработкой (Dev) и эксплуатацией (Ops), ведущие к задержкам и проблемам с внедрением.

5.3. *Разрешение*: становление культуры и практик DevOps, автоматизация процессов CI/CD (Continuous Integration/Continuous Delivery), микросервисная архитектура.

6. *2020-е: ИИ, low-code и автоматизация (текущий Тезис)*

6.1. *Состояние*: активное внедрение ИИ и машинного обучения в ПО и в инструменты разработки (например, GitHub Copilot), рост платформ low-code/no-code, фокус на Everything-as-a-Service.

6.2. *Противоречие*: растущая потребность в ускорении разработки и нехватка квалифицированных разработчиков противопоставлена сложности и потенциальной непрозрачности ИИ-генерируемого кода, риски качества и безопасности.

6.3. *Разрешение (прогнозируемый Синтез)*: развитие инструментов верификации и валидации ИИ-генерируемого кода, методик объяснимого ИИ (Explainable AI), усиление роли человека в контроле качества при высокой автоматизации.

Проведем анализ эволюции методологий разработки ПО (Waterfall → Agile → DevOps) через призму диалектики.

Этот процесс является классическим примером закона отрицание отрицания.

1. *Тезис: Waterfall (Водопадная модель)*

1.1. *Характеристики*: линейная последовательность этапов (требования → проектирование → реализация → тестирование → внедрение); ценности: строгий порядок, предсказуемость, документированность, контроль.

1.2. *Внутренние противоречия*: низкая гибкость к изменениям требований, длительный цикл обратной связи, эффект «большого взрыва» при релизе, проблемы при работе над сложными инновационными проектами.

2. *Антитезис: Agile (Гибкие методологии)*

2.1. *Характеристики*: итеративная и инкрементальная разработка, короткие циклы, тесное взаимодействие с заказчиком, фокус на работающем продукте и быстрой адаптации к изменениям; ценности: люди и взаимодействие важнее процессов и инструментов, работающий продукт важнее документации, сотрудничество с заказчиком важнее согласования условий контракта, готовность к изменениям важнее следования первоначальному плану.

2.2. *Диалектическое отрицание Waterfall*: Agile отрицает жесткость, линейность и чрезмерную формализацию Waterfall.

2.3. *Новые противоречия*: сложности в масштабировании на крупные проекты, высокие требования к квалификации и культуре команды, потенциальный разрыв между быстрой разработкой (Dev) и стабильной эксплуатацией (Ops).

3. *Синтез (Снятие): DevOps (Разработка и Эксплуатация)*

3.1. *Характеристики*: культура, практики и инструменты, направленные на автоматизацию и интеграцию процессов разработки ПО и его эксплуатации; фокус на CI/CD, инфраструктуре как коде (Infrastructure as Code), мониторинге, коллаборации.

3.2. *Диалектическое снятие*: DevOps не просто отрицает Agile, а интегрирует его принципы (скорость, гибкость) с потребностями эксплуатации (стабильность, надежность). Он «снимает» противоречие между Dev и Ops.

3.2.1. *Сохраняет*: ценности Agile (итеративность, фокус на быстрой поставке ценности).

3.2.2. *Отрицает*: разделенность и конфликт между разработкой и эксплуатацией.

3.2.3. *Поднимает на новый уровень*: весь процесс поставки ПО становится более быстрым, надежным и эффективным.

В философском отношении этот переход нельзя понимать лишь как смену управленческой парадигмы. Водопадная модель выражала логику индустриального производства, в рамках которой проект рассматривался как заранее заданная совокупность требований, подлежащих последовательной реализации. Ее жесткость была связана с исторически характерной установкой на предсказуемость процесса и нормативно заданную завершенность результата.

Agile, напротив, сформировался как ответ на ограничения этой модели в условиях возрастающей изменчивости цифровой среды. Он поставил под сомнение возможность полного предварительного определения требований и тем самым сместил акцент с детального планирования на адаптивность и итеративное уточнение решений. Вместе с тем в ранних формах Agile приоритет гибкости на уровне отдельной команды нередко приводил к новому противоречию – рассогласованию между скоростью разработки и устойчивостью эксплуатации.

DevOps в этом контексте выступает не как компромисс, а как форма снятия указанного противоречия. Он сохраняет принцип итеративности и быстрой обратной связи, но интегрирует его в более широкую систему непрерывной поставки и эксплуатации. Тем самым происходит переход к более конкретной организационной целостности, в которой разработка и эксплуатация рассматриваются как взаимно определяющие моменты единого процесса.

С философской точки зрения это движение можно интерпретировать как переход от внешнего управления сложностью к формированию механизмов внутренней саморегуляции системы. Индустрия разработки ПО постепенно институционализирует собственную рефлексию: инструменты мониторинга, автоматизированного тестирования и непрерывной интеграции становятся не внешними надстройками, а структурными элементами самой логики производства программного продукта.

Обратимся к диалектике развития архитектурных парадигм программного обеспечения.

Параллельно с эволюцией методологий разработки, не менее значимые диалектические процессы наблюдаются в развитии архитектурных парадигм программного обеспечения. Смена доминирующих архитектурных стилей также может быть проанализирована через призму гегелевской триады, где каждый последующий подход «снимает» противоречия предыдущего, сохраняя его рациональные зерна и поднимая систему на новый качественный уровень.

1. *Тезис: монолитная архитектура*

1.1. *Характеристики*: на ранних этапах развития индустрии, когда программные системы оставались сравнительно компактными, монолитная архитектура была вполне рациональным решением; тесная связанность компонентов и развертывание приложения как единого целого упрощали разработку, тестирование и ввод системы в эксплуатацию – для своего этапа развития такой подход был не только технически оправдан, но и организационно удобен.

1.2. *Внутренние противоречия*: по мере роста масштаба и усложнения приложений преимущества монолита постепенно превращались в ограничения, иными словами, количественное расширение системы сопровождалось нарастанием качественных проблем.

1.2.1. *Сложность масштабирования*: поскольку приложение функционировало как единое целое, масштабировать отдельные функциональные блоки независимо друг от друга было затруднительно. В результате увеличение нагрузки на один компонент нередко требовало избыточного расширения ресурсов всей системы.

1.2.2. *Низкая отказоустойчивость*: высокая связанность компонентов означала, что сбой в одной части приложения мог повлиять на работоспособность системы в целом.

1.2.3. *Трудности с обновлением и внедрением новых технологий*: даже локальные изменения часто требовали пересборки и повторного развертывания всего приложения, что замедляло цикл разработки и повышало риски; кроме того, единый технологический стек ограничивал возможность точечного внедрения новых решений.

1.2.4. *Снижение скорости разработки и рост сложности координации*: по мере роста кодовой базы и увеличения числа разработчиков возрастали издержки согласования; командам становилось сложнее понимать структуру системы, координировать изменения и избегать конфликтов в коде.

2. *Антитезис: сервис-ориентированная архитектура (SOA) и ранние распределенные системы*

2.1. *Характеристики*: ответом на ограничения монолитного подхода стало развитие идей декомпозиции приложения на более мелкие и относительно слабо связанные сервисы; в рамках архитектуры «SOA» предлагались стандартизированные механизмы взаимодействия между сервисами, нередко с использованием корпоративной сервисной шины (Enterprise Service Bus, ESB); благодаря такому подходу повышалась модульность системы и появлялась возможность повторного использования компонентов.

2.2. *Диалектическое отрицание монолита*: в этом смысле SOA стала отрицанием монолитной логики: она преодолевала жесткую связанность и неделимость приложения, открывая возможность для большей гибкости и более самостоятельного развития отдельных частей системы.

2.3. *Новые противоречия*: несмотря на прогресс, SOA и ранние распределенные системы породили собственные сложности.

2.3.1. *Высокая сложность управления и координации сервисов*: ESB мог становиться «узким горлышком» и центральной точкой отказа, усложняя систему.

2.3.2. *Тяжеловесность протоколов взаимодействия* (например, SOAP) и избыточная сложность стандартов.

2.3.3. *Недостаточная гранулярность и автономность сервисов*: сервисы в SOA часто оставались достаточно крупными и не всегда обладали полной независимостью в развертывании и управлении данными.

3. *Синтез (Снятие): Микросервисная архитектура*

3.1. *Характеристики*: микросервисная архитектура представляет собой дальнейшее развитие идей SOA, но с акцентом на создание очень маленьких, полностью независимых и

автономных сервисов; каждый сервис решает одну конкретную бизнес-задачу, имеет собственную, независимую базу данных (принцип database per service), разрабатывается, тестируется, развертывается и масштабируется независимо от других.

3.2. *Диалектическое снятие*: микросервисы «снимают» противоречия как монолитов, так и многих реализаций SOA:

3.2.1. *Сохраняют*: идею декомпозиции, сервис-ориентированности и слабой связанности от SOA.

3.2.2. *Отрицают*: жесткую связанность, неделимость и технологическую унификацию монолита, а также тяжеловесность, централизованность (например, отказ от центрального ESB в пользу «умных» конечных точек и «глупых» каналов – smart endpoints and dumb pipes) и недостаточную гранулярность некоторых подходов SOA.

3.2.3. *Поднимают на новый уровень*: обеспечивают высокую степень технологической гетерогенности (каждый сервис может быть написан на подходящем для него языке/технологии), максимальную независимость команд разработки, улучшенную масштабируемость отдельных функций и повышенную отказоустойчивость системы в целом; этот подход отлично коррелирует с практиками Agile и DevOps, позволяя реализовывать чаще и независимые релизы.

3.3. *Новые противоречия (задел для будущего развития)*: распределенность микросервисной архитектуры, в свою очередь, порождает новые вызовы: сложность управления большим количеством сервисов (мониторинг, журналирование, обнаружение сервисов, трассировка запросов), обеспечение согласованности данных между сервисами (distributed transactions), необходимость в сложных платформах оркестрации контейнеров (например, Kubernetes) и сервис-мешах (service mesh) для управления межсервисным взаимодействием; эти новые противоречия уже побуждают к поиску следующих архитектурных решений – в частности, к развитию бессерверных вычислений (serverless), событийно-ориентированных архитектур (event-driven architectures), а также новых средств автоматизации и управления сложными распределенными системами.

Развитие архитектурных парадигм в индустрии ПО подчиняется сходной логике. Монолитная архитектура была эффективна на ранних этапах, когда система еще не достигала высокой степени сложности и могла развиваться как единое целое. Переход к сервис-ориентированному подходу означал попытку расчленить эту целостность на отдельные функциональные блоки, однако при этом сохранялись элементы централизованного управления и координации. Микросервисная архитектура стала следующим шагом в этом развитии: она опирается на большую автономию компонентов, но одновременно требует устойчивых механизмов их взаимодействия через стандартизированные интерфейсы. В результате архитектурная эволюция идет не просто по линии дробления системы, а по линии формирования более сложной и гибкой структуры, способной сочетать функциональную дифференциацию с общей связностью.

5. Обсуждение

Применение диалектического метода Гегеля дало возможность по-новому посмотреть на эволюцию индустрии разработки программного обеспечения и выделить в ней устойчивые линии развития. Рассмотрение исторических этапов и смены методологий (Waterfall → Agile → DevOps) показывает, что переход от одной формы организации разработки к другой был

связан не со случайной сменой управленческой моды, а с накоплением внутренних противоречий внутри самой практики разработки. По мере их обострения количественные изменения приводили к качественным сдвигам, а новые формы не просто заменяли прежние, а в определенной мере перерабатывали и сохраняли их отдельные элементы.

Полученные результаты позволяют интерпретировать смену методологий как закономерный процесс. Каждая следующая модель складывалась как ответ на ограничения предыдущей. Так, Agile сформировался на фоне недостаточной гибкости Waterfall, тогда как DevOps вырос из необходимости преодолеть разрыв между ускорением разработки и требованиями стабильной эксплуатации. В этом смысле последовательность Waterfall, Agile и DevOps может быть прочитана через диалектическую логику: новая форма возникает из противоречий прежней, не устраняя их механически, а преобразуя сам способ организации процесса.

В данной работе основной акцент сделан на философской реконструкции этих переходов и на попытке их формализации. Вместе с тем между философским уровнем анализа и математическим описанием может быть введен дополнительный экономический уровень интерпретации. Он позволяет связать выявленные закономерности с измеримыми характеристиками индустрии разработки ПО и тем самым сделать предложенную модель более содержательной для последующих исследований.

Ключевым экономическим фактором, лежащим в основе выявленных противоречий, выступает неопределенность требований к программным системам. В условиях высокой изменчивости пользовательских предпочтений, технологических стандартов и конкурентной среды требования к продукту не могут быть полностью специфицированы заранее. Это делает невозможным полное предварительное планирование и порождает систематическое расхождение между проектными ожиданиями и фактической динамикой разработки.

Экономической формой проявления данной неопределенности является рост транзакционных издержек – издержек координации, согласования изменений, перераспределения ответственности и переработки уже реализованных компонентов. По мере усложнения систем и увеличения числа участников проекта предельные координационные издержки возрастают нелинейно, что может быть интерпретировано как накопление индекса диалектического противоречия $\Delta(t)$, введенного в модели.

С этой точки зрения переходы между методологиями разработки (например, от каскадных моделей к гибким практикам, а затем к DevOps-интеграции) могут рассматриваться как институциональные механизмы снижения транзакционных издержек в условиях устойчивой неопределенности среды. Аналогично архитектурные трансформации отражают поиск организационных форм, позволяющих сохранить инвестиции в программные активы при одновременном снижении издержек модификации и переключения.

Дальнейшее развитие модели может быть связано с более явной формализацией экономических интересов агентов – разработчиков, менеджмента, владельцев продукта и пользователей, а также с построением агент-ориентированных моделей, учитывающих различие их горизонтов планирования и структур стимулов. Это позволило бы связать философскую реконструкцию развития отрасли с инструментарием институциональной и эволюционной экономики и тем самым расширить возможности количественного анализа.

6. Вклад в теорию

Теоретическое значение проведенного исследования связано прежде всего с попыткой расширить методологические возможности экономико-математического анализа технологических отраслей. В работе философский подход не противопоставляется инструментальным методам, а рассматривается как основание для их более содержательного применения. Такой ракурс позволяет показать, что диалектические принципы могут использоваться не только на уровне общей интерпретации, но и при анализе сложных экономических систем, для которых важны внутренние противоречия, переходные состояния и качественные сдвиги.

Существенным результатом исследования является и сама трактовка технологических инноваций. В рамках предложенного подхода они рассматриваются не как последовательный линейный прогресс, а как процесс, в котором развитие идет через накопление противоречий, их обострение и последующее возникновение новых организационных и технологических форм. Такая перспектива позволяет точнее описывать логику изменений в индустрии программного обеспечения, чем модели, сводящие развитие только к поступательному наращиванию эффективности.

Отдельное значение имеет обращение к идеям Э.В. Ильенкова о восхождении от абстрактного к конкретному (Ильенков, 1960). В контексте настоящего исследования этот принцип дает возможность перейти от философских категорий к экономико-математическим моделям и измеряемым параметрам индустрии ПО. Именно благодаря такому переходу становится возможной содержательная связь между философской методологией и инструментальными средствами анализа.

7. Практическая ценность

1. Предложенный методологический подход может быть использован ИТ-компаниями для стратегического планирования, анализа текущих вызовов и прогнозирования будущих трендов в технологиях и методологиях разработки.

2. Понимание диалектической природы развития помогает более осознанно подходить к выбору и адаптации методологий разработки, предвидя возможные противоречия и пути их разрешения.

3. Модели, основанные на диалектических принципах (пусть и упрощенно формализованные), могут служить инструментом для оценки последствий внедрения новых технологий и организационных изменений в контексте цифровой трансформации бизнеса и государственных структур.

Сделаем диалектический прогноз следующего витка развития индустрии разработки ПО (с 2025 г.).

Опираясь на текущий этап (синтез ИИ, low-code, автоматизации для решения проблемы дефицита ресурсов и ускорения разработки), можно спрогнозировать следующий виток диалектического развития – текущий синтез, антитезис, новый синтез.

1. Текущий синтез (2020-е): внедрение ИИ, low-code/no-code для ускорения разработки и компенсации дефицита кадров.

2. Новое противоречие (антитезис).

2.1. ИИ-генерируемый код может быть сложным, непрозрачным («черный ящик»), содержать скрытые дефекты или уязвимости.

2.2. Возникают трудности с отладкой, поддержкой, масштабированием и обеспечением долгосрочной эволюции систем, созданных с активным участием ИИ.

2.3. Усиливаются требования к верификации, валидации, контролю качества и безопасности ПО, при этом сами инструменты анализа усложняются.

2.4. Противоречие между скоростью генерации кода ИИ и способностью человека осмысленно контролировать и управлять этим процессом.

3. Предполагаемое разрешение (новый синтез).

3.1. Развитие продвинутых инструментов для верификации, валидации и тестирования ИИ-генерируемого кода, включая формальные методы и специализированные анализаторы.

3.2. Становление и широкое внедрение методик объяснимого ИИ (Explainable AI) в разработке ПО, позволяющих понимать логику принятия решений ИИ-инструментами и структуру генерируемого ими кода.

3.3. Разработка гибридных методологий, где ИИ берет на себя рутинные и хорошо формализуемые задачи, а человек концентрируется на творческих аспектах, архитектуре, критическом контроле качества и принятии стратегических решений.

3.4. Формирование новых ролей в ИТ-командах, например, «AI Code Auditor» или «AI Systems Maintainer».

Модельные параметры для прогноза приведены в табл. 1.

Таблица 1

Модельные параметры прогноза

Параметр	Текущий уровень (2020-е)	Ожидаемый уровень (2025+)
Автоматизация генерации кода (ИИ)	Высокая для отдельных модулей/задач	Очень высокая, комплексная генерация систем
Прозрачность и объяснимость ИИ-кода	Низкая/средняя, эффект «черного ящика»	Средняя/высокая, за счет развития ХАИ (Explainable AI) и специализированных инструментов анализа
Сложность поддержки ИИ-кода	Средняя, растущая	Высокая, требующая новых подходов и инструментов
Инструменты контроля качества ИИ-кода	Начальный этап развития	Развитые, специализированные системы верификации, валидации и этического аудита ИИ-кода
Роль человека в цикле разработки с ИИ	Контроль на высоком уровне, постановка задач	Критический надзор, валидация, этическая оценка, управление сложными ИИ-системами

Источник: составлено автором.

8. Ограничения исследования

1. Сложность полной и строгой формализации философских категорий диалектики в математических моделях. Предложенные модели являются упрощениями.
2. Доступность и качество детализированных исторических данных по экономическим показателям ИТ-проектов могут быть ограничены.
3. Субъективность в интерпретации исторических событий и выделении «противоречий» и «синтезов», хотя стремление к объективности достигается через анализ авторитетных источников и отраслевых отчетов.

9. Рекомендации для будущих исследований

1. Разработка и верификация количественных моделей: построение конкретных системно-динамических или агентных моделей на основе предложенных концепций, их калибровка и верификация на реальных данных индустрии ПО.
2. Эмпирические исследования: проведение кейс-стади конкретных ИТ-компаний или крупных проектов для выявления диалектических закономерностей в их развитии.
3. Междисциплинарные исследования: углубление сотрудничества между философами, экономистами-математиками и специалистами по программной инженерии для совместной разработки и уточнения методологии.
4. Анализ влияния ИИ на диалектику развития: исследование того, как сам ИИ (возможно, в будущем обладающий элементами диалектического «мышления») может изменять характер эволюции индустрии ПО.
5. Применение подхода к другим цифровым областям: адаптация предложенной методологии для анализа развития других быстрорастущих секторов цифровой экономики (финтех, электронная коммерция, биотехнологии).

10. Предмет обсуждения для научной дискуссии

1. Каковы границы применимости диалектического метода в экономико-математическом моделировании технологического развития? Где он перестает быть эвристическим инструментом и становится лишь метафорой?
2. Возможно ли создать универсальную формализованную диалектическую модель, способную с высокой точностью прогнозировать качественные скачки в развитии сложных социотехнических систем, таких как индустрия ПО?
3. Какие конкретные метрики и индикаторы могут наиболее адекватно отражать наличие и степень обострения «диалектических противоречий» в ИТ-проектах и в индустрии в целом?
4. Как концепция «снятия» (Aufhebung) может быть интерпретирована и измерена в контексте оценки эффективности внедрения новых методологий или технологий?

Заключение

Проведенный анализ позволяет сделать вывод о том, что диалектический метод Г.В.Ф. Гегеля действительно обладает серьезным эвристическим потенциалом применительно

к исследованию индустрии разработки программного обеспечения. Его продуктивность проявляется прежде всего в возможности рассматривать развитие этой индустрии не как линейную последовательность технологических новшеств, а как процесс, в котором новые формы возникают из внутренних противоречий предшествующих состояний. Именно такая оптика позволяет увидеть в смене методологий и архитектур не набор разрозненных инноваций, а внутренне связанное движение.

Важным для данной работы стало то, что диалектический подход использовался не только на уровне общей философской интерпретации, но и как основание для дальнейшей аналитической конкретизации. В этом контексте продуктивными оказались и те схемы, которые были разработаны позднее в марксистской традиции, прежде всего у Ф. Энгельса: развитие через противоречия, переход количественных изменений в качественные, отрицание отрицания. В рамках настоящего исследования они рассматривались не как самостоятельная догматическая система, а как удобные аналитические конструкции, позволяющие описывать исторические сдвиги, кризисы прежних форм и возникновение новых организационных решений в индустрии ПО.

Рассмотрение перехода от Waterfall к Agile, а затем к DevOps, так же, как и смены архитектурных моделей от монолита к SOA и далее к микросервисам, показало, что каждая новая форма складывается не на пустом месте. Она появляется в ответ на ограничения предыдущей, перерабатывает их и одновременно сохраняет те элементы, которые сохраняют свою функциональную значимость в новых условиях. Благодаря этому диалектический метод оказывается полезен не только для ретроспективного описания уже состоявшихся изменений, но и для объяснения самой логики их возникновения.

Принцип восхождения от абстрактного к конкретному, разработанный Э.В. Ильенковым, оказался особенно важен для связывания философского и инструментального уровней анализа. Именно через него стало возможным перейти от общих категорий развития, противоречия и снятия к более конкретным моделям, параметрам и аналитическим схемам, пригодным для дальнейшей формализации. Тем самым философская методология в работе выступает не внешним комментарием по отношению к технологической реальности, а рамкой, внутри которой можно строить экономико-математические и имитационные модели.

Научное значение проведенного исследования связано, прежде всего, с попыткой соединить философский анализ, проблематику программной инженерии и инструменты экономико-математического моделирования в единой исследовательской логике. Такой междисциплинарный ход позволяет расширить методологический горизонт исследований цифровой экономики и показать, что диалектические категории могут быть использованы не только в теоретическом рассуждении, но и в построении моделей сложных социотехнических систем.

Практическое значение работы состоит в том, что предложенный подход может служить основой для более осмысленного анализа текущих изменений в индустрии разработки ПО. Он дает возможность рассматривать смену методологий, архитектур и инструментов не как набор частных управленческих решений, а как проявление более глубоких структурных сдвигов. В этом смысле диалектическая перспектива может быть полезна и для исследовательских центров, и для ИТ-компаний, которым важно не только фиксировать текущие тренды, но и понимать внутренние источники будущих изменений.

В целом проведенное исследование следует рассматривать как шаг к дальнейшей разработке моделей, способных соединить философскую реконструкцию развития индустрии ПО

с эмпирическим и количественным анализом. Перспективы такого подхода особенно заметны в условиях ускоряющегося внедрения искусственного интеллекта, автоматизации и новых организационных форм разработки, когда вопрос о внутренних противоречиях технологического развития приобретает не только теоретическое, но и вполне прикладное значение.

Список литературы

- Абдеев Р.Ф. Философия информационной цивилизации. М.: Владос, 1994. 336 с.
- Алиев Б.Н. Генетические алгоритмы как инструмент формирования и эволюции торговых стратегий на рынке ценных бумаг // Финансы и кредит. 2024. Т. 30. № 4 (844). С. 851–872. <https://doi.org/10.24891/fc.30.4.851>.
- Беркова О.В. Использование диалектического метода в современной экономической теории // Экономический рост Республики Беларусь: глобализация, инновационность, устойчивость: Материалы XV Международной научно-практической конференции, Минск, 19–20 мая 2022 года. Минск: Белорусский государственный экономический университет, 2022. С. 479–480.
- Гегель Г.В.Ф. Наука логики: В 3 т. Т. 1. М.: «Мысль», 1970. 501 с.
- Емельяненко В.Д., Яненко Е.М. Человек и Интернет: диалектика знаний и информации // Философские проблемы информационных технологий и киберпространства. 2021. № 2. С. 4–21. <https://doi.org/10.17726/phillIT.2021.2.1>.
- Ильенков Э.В. Диалектика абстрактного и конкретного в научно-теоретическом мышлении. М.: Наука, 1960. 464 с.
- Мыльник В.В. Диалектика развития автоматизации промышленного производства на различных ее этапах // Организатор производства. 2009. № 4.
- Энгельс Ф. Диалектика природы // Маркс К., Энгельс Ф. Соч. 2-е изд. Т. 20. М.: Политиздат, 1961. С. 339–570.
- Almeida F., Simões J., Lopes S. Exploring the Benefits of Combining DevOps and Agile // Future Internet. 2022. Vol. 14 (2). P. 63. <https://doi.org/10.3390/fi14020063>.
- Johnson L. A view from the 1960s: How the software industry began // IEEE Annals of the History of Computing. 1998. Vol. 20 (1). P. 36–42.
- Joseph C.P. Philosophical Methodology, Technologies, and the Transformation of Knowledge // Philosophy and Technology. 1996. Vol. 1 (3–4).
- Kirongo A.C., Otieno O. Research Philosophy, Design and Methodologies: A Systematic Review of Research Paradigms in Information Technology // Global Scientific Journal. 2020. Vol. 8 (5).
- Lee J.A.N., John B., George R. FORTRAN Session // History of Programming Languages. Academic Press. 1981. P. 25–74.
- Randell B. The 1968/69 NATO software engineering reports // History of software engineering. 1996. Vol. 37.
- Robinson B. Dialectics and modelling in information systems // Systemist 20 (Special Edition). 1998. P. 208–220
- Swaleh M., Frankline W. IT Philosophy: Philosophical Paradigms in Information Technology Research // International Journal of Research and Innovation in Applied Science. 2025. Vol. 10 (1). P. 259–268. <https://doi.org/10.51584/IJRIAS.2025.1001021>.

Grand View Research. Software market size, share and trends // Industry report. 2025: URL: <https://www.grandviewresearch.com/industry-analysis/software-market-report> (дата обращения: 20.09.2025).

IoT Analytics. The industrial software market landscape: 7 key statistics going into 2025: URL: <https://iot-analytics.com/industrial-software-market-landscape/> (дата обращения: 27.09.2025).

Wernick P., Christianson B., Loomes M.J., Shearer D.W. A Dialectical Basis for Software Development Tool Building // AMCIS 2000 Proceedings. 2000. <http://aisel.aisnet.org/amcis2000/31> (дата обращения: 28.12.2025).

Приложение

Python код демонстративной вычислительной модели диалектической метамодели

```

"""
Демонстрационная вычислительная модель диалектической метамодели
-----

Этот скрипт реализует упрощенную вычислительную иллюстрацию
диалектической метамодели, описывающей накопление внутренних
противоречий в системе разработки программного обеспечения.

ВАЖНО:
- Это НЕ эмпирическая или прогнозная модель.
- Это концептуальный вычислительный эксперимент.
- Все параметры являются иллюстративными и не калиброваны по реальным данным.

Цель состоит в том, чтобы продемонстрировать:
1) количественное накопление противоречий,
2) качественный переход на основе порогового значения (диалектический скачок),
3) реконфигурацию параметров системы после перехода.
«»"

Переведено с помощью DeepL.com (бесплатная версия)
"""

import matplotlib.pyplot as plt

import numpy as np

# -----
# 1. Simulation parameters
# -----

T = 150          # total number of time steps
time = np.arange(T)

# Target (normative) values
V_required = 1.0    # required delivery speed
S_target = 1.0      # target quality level

# Critical threshold of contradiction
delta_critical = 0.30

# Weights of contradiction components
w_speed = 0.6
w_quality = 0.4

# -----
# 2. State variables
# -----

V = np.zeros(T)    # delivery speed
S = np.zeros(T)    # quality / stability
A = np.zeros(T)    # level of automation
Delta = np.zeros(T) # dialectical contradiction index

# -----
# 3. Initial conditions (Waterfall-like state)
# -----

V[0] = 0.30        # low speed
S[0] = 0.90        # high initial quality
A[0] = 0.20        # low automation

phase_transition_time = None

# -----

```

```
# 4. Simulation loop
# -----

for t in range(1, T):

    # 4.1 Gradual growth of automation (quantitative change)
    A[t] = A[t - 1] + 0.004

    # 4.2 Delivery speed grows as a function of automation
    V[t] = V[t - 1] + 0.03 * A[t]

    # 4.3 Quality degradation due to excessive speed
    # This models the internal contradiction between speed and quality
    if V[t] > 0.65:
        S[t] = S[t - 1] - 0.02 * (V[t] - 0.65)
    else:
        S[t] = S[t - 1]

    # 4.4 Boundary constraints
    V[t] = min(V[t], 1.25)
    S[t] = max(min(S[t], 1.0), 0.3)

    # 4.5 Dialectical contradiction index
    # Measures the deviation from desired system state
    Delta[t] = (
        w_speed * (V_required - V[t]) ** 2 +
        w_quality * (S_target - S[t]) ** 2
    )

    # 4.6 Qualitative transition (dialectical leap)
    if Delta[t] >= delta_critical and phase_transition_time is None:

        phase_transition_time = t

        # Dialectical sublation (Aufhebung):
        # partial negation of previous dynamics with preservation
        V[t] *= 0.85      # temporary reduction of speed
        S[t] += 0.15      # restoration of quality
        S[t] = min(S[t], 1.0)

# -----
# 5. Output variables
# -----

# The following arrays represent the system dynamics:
# - V[t] : delivery speed over time
# - S[t] : quality level over time
# - A[t] : automation level over time
# - Delta[t] : accumulated dialectical contradiction
# - phase_transition_time : moment of qualitative transition

# --- График 1: Скорость поставки ---
plt.figure()
plt.plot(time, V)
if phase_transition_time is not None:
    plt.axvline(phase_transition_time, linestyle='--')
plt.xlabel("Время")
plt.ylabel("Скорость поставки V(t)")
plt.title("Динамика скорости поставки")
plt.savefig('fДинамика_скорости_поставки.png', dpi=600)
plt.show()

# --- График 2: Качество ---
plt.figure()
plt.plot(time, S)
if phase_transition_time is not None:
    plt.axvline(phase_transition_time, linestyle='--')
plt.xlabel("Время")
plt.ylabel("Уровень качества S(t)")
plt.title("Динамика качества")
plt.savefig('fДинамика_качества', dpi=600)
```

```
plt.show()

# --- График 3: Диалектическое противоречие ---
plt.figure()
plt.plot(time, Delta)
plt.axhline(delta_critical, linestyle='--')
if phase_transition_time is not None:
    plt.axvline(phase_transition_time, linestyle='--')
plt.xlabel("Время")
plt.ylabel("Индекс диалектического противоречия  $\Delta(t)$ ")
plt.title("Накопление диалектического противоречия")
plt.savefig('Накопление_диалектического_противоречия', dpi=600)
```

Mathematical and Statistical methods

HEGEL'S DIALECTICAL METHOD IN ANALYZING THE EVOLUTION OF THE SOFTWARE DEVELOPMENT INDUSTRY

Beilak A. Aliev

*Lomonosov Moscow State University, Faculty of Economics
(Moscow, Russia)*

Abstract

The article presents a methodological approach to analysing the evolution of the software development industry based on G.W.F. Hegel's dialectical method. The central analytical principle is the idea of development through internal contradictions, reflected in the logic of the formation and removal (Aufhebung) of opposites as a key mechanism for the self-development of concepts and systems. Within the framework of the study, the Hegelian approach correlates with the provisions subsequently formulated in Marxist philosophy as dialectical laws – development through contradictions (instead of the opposition of ‘unity and struggle of opposites’), the transition from quantitative to qualitative changes, and the negation of negation – interpreted in this work as instrumental models for describing the stages of transformation of technological and organisational paradigms in the field of software development. Particular attention is paid to the application of the principle of movement from the abstract to the concrete, including in the interpretation of E.V. Ilyenkov, for the formalisation of historical and logical analysis of transitions between development methodologies (Waterfall, Agile, DevOps) and modern forms, conditioned by the spread of generative neural networks, low-code/no-code solutions, and intelligent automation. The scientific and practical value of applying the dialectical method in the context of analysing high-tech industries is substantiated. The article proposes a metamodel of dialectical analysis, intended for subsequent implementation in system dynamics and agent-based modelling tools. The work is philosophical and methodological in nature and is focused on forming an operational basis for further empirical and simulation studies. The limitations of the proposed approach and directions for further research are also considered, including the potential for predicting future changes in the software industry under the influence of artificial intelligence systems.

Keywords: dialectical method, Hegel, software development, evolution of the software industry, software development methodologies, Waterfall, Agile, DevOps, computer science, LLM, AI, low-code, no-code.

JEL: C18, C63, C87.

For citation: Aliev, B.N. (2026) Hegel's Dialectical Method in Analyzing the Evolution of the Software Development Industry. Scientific Research of Faculty of Economics. Electronic Journal, vol. 18, no. 3, pp. 9-36. DOI: 10.38050/2078-3809-2026-18-3-9-36.

References

- Abdeev R.F. *Filosofiya informatsionnoy tsivilizatsii*. M.: Vldos, 1994. 336 p. (In Russ.).
- Aliev B.N. Geneticheskie algoritmy kak instrument formirovaniya i evolyutsii torgovykh strategiy na rynke tsennykh bumag. *Finansy i kredit*. 2024. Vol. 30. №No. 4 (844). P. 851–872. <https://doi.org/10.24891/fc.30.4.851>. (In Russ.).
- Berkova O.V. Ispol'zovanie dialekticheskogo metoda v sovremennoy ekonomicheskoy teorii. *Ekonomicheskiy rost Respubliki Belarus': globalizatsiya, innovatsionnost', ustoychivost'*: Materialy XV Mezhdunarodnoy nauchno-prakticheskoy konferentsii, Minsk, 19–20 maya 2022 goda. Minsk: Belorusskiy gosudarstvennyy ekonomicheskiy universitet, 2022. P. 479–480. (In Russ.).
- Gegel' G.V.F. *Nauka logiki: V 3 t. T. 1. M.: «Mysl'», 1970. 501 p. (In Russ.).*
- Emel'yanenko V.D., Yanenko E.M. Chelovek i Internet: dialektika znaniy i informatsii. *Filosofskie problemy informatsionnykh tekhnologiy i kiberprostranstva*. 2021. № 2. S. 4–21. <https://doi.org/10.17726/philIT.2021.2.1>. (In Russ.).
- Il'enkov E.V. *Dialektika abstraktnogo i konkretnogo v nauchno-teoreticheskom myshlenii*. M.: Nauka, 1960. 464 s. (In Russ.).
- Myl'nik V.V. Dialektika razvitiya avtomatizatsii promyshlennogo proizvodstva na razlichnykh ee etapakh. *Organizator proizvodstva*. 2009. № 4. (In Russ.).
- Engel's F. *Dialektika prirody*. Marks K., Engel's F. Soch. 2-e izd. T. 20. M.: Politizdat, 1961. S. 339–570. (In Russ.).
- Almeida F., Simões J., Lopes S. Exploring the Benefits of Combining DevOps and Agile. *Future Internet*. 2022. Vol. 14 (2). P. 63. <https://doi.org/10.3390/fi14020063>.
- Johnson L. A view from the 1960s: How the software industry began. *IEEE Annals of the History of Computing*. 1998. Vol. 20 (1). P. 36–42.
- Joseph C.P. Philosophical Methodology, Technologies, and the Transformation of Knowledge. *Philosophy and Technology*. 1996. Vol. 1 (3–4).
- Kirongo A.C., Otieno O. Research Philosophy, Design and Methodologies: A Systematic Review of Research Paradigms in Information Technology. *Global Scientific Journal*. 2020. Vol. 8 (5).
- Lee J.A.N., John B., George R. *FORTTRAN Session. History of Programming Languages*. Academic Press. 1981. P. 25–74.
- Randell B. The 1968/69 NATO software engineering reports. *History of software engineering*. 1996. Vol. 37.
- Robinson B. Dialectics and modelling in information systems. *Systemist 20 (Special Edition)*. 1998. P. 208–220
- Swaleh M., Frankline W. IT Philosophy: Philosophical Paradigms in Information Technology Research. *International Journal of Research and Innovation in Applied Science*. 2025. Vol. 10 (1). P. 259–268. Available at: <https://doi.org/10.51584/IJRIAS.2025.1001021>.
- Grand View Research. Software market size, share and trends. Industry report. 2025: Available at: <https://www.grandviewresearch.com/industry-analysis/software-market-report> (accessed: 20.09.2025).
- IoT Analytics. The industrial software market landscape: 7 key statistics going into 2025: Available at: <https://iot-analytics.com/industrial-software-market-landscape/> (accessed: 27.09.2025).
- Wernick P., Christianson B., Loomes M.J., Shearer D.W. A Dialectical Basis for Software Development Tool Building. *AMCIS 2000 Proceedings*. 2000. Available at: <http://aisel.aisnet.org/amcis2000/31> (accessed: 28.12.2025).