РОЛЬ АРХИТЕКТУРНЫХ РЕШЕНИЙ В СНИЖЕНИИ ТЕХНИЧЕСКОГО ДОЛГА

Мусатов Антон Олегович,
инженер-программист,
Digitail Inc г. Ереван, Армения

Аннотация. В статье рассматривается проблема технического долга как ключевого фактора, влияющего на качество и сопровождаемость программных систем. Подчёркивается, что архитектурный технический долг является наиболее значимым по последствиям, поскольку напрямую влияет на масштабируемость, отказоустойчивость и гибкость ПО. Представлены основные виды технического долга и методы его оценки. Особое внимание уделяется роли архитектурных решений в управлении техническим долгом: рассмотрены принципы модульности, слабой связанности, высокой когезии, явных границ контекста и эволюционной архитектуры. Приведены практические примеры из индустрии и результаты исследований, подтверждающие значимость архитектурных подходов в снижении издержек сопровождения и ускорении разработки. Отмечены проблемы реализации архитектурных решений. Сформулированы практические рекомендации для разработчиков и архитекторов, а также очерчены перспективы дальнейших исследований.

Ключевые слова: технический долг, архитектурный долг, архитектурные решения, модульность, когезия, слабая связанность, DevOps, CI/CD, ISO/IEC 5055, ATDM, SonarQube, эволюционная архитектура.

Актуальность исследования

Проблема технического долга усугубляется в современных организациях. Быстрый выпуск новых функций зачастую происходит за счёт внесения компромиссов в проектирование, что усложняет сопровождение и развитие продуктов в долгосрочной перспективе. Исследования показывают, что технический долг – это не просто отложенные правки, а накопление затрат и сниженный темп разработки со временем.

Особое значение в данной проблематике приобретают архитектурные дефекты. Отчёты обращают внимание на масштабы ущерба: ожидаемые потери от технического долга составляют порядка 1,52 трлн долларов в год. Более половины организаций тратят значительную часть (более 25%) ИТ-бюджета на устранение этого долга, особенно если он связан с монолитной архитектурой, которая почти в 2,1 раза более подвержена проблемам масштабируемости, отказоустойчивости и скорости разработки по сравнению с микросервисами [2].

Архитектурная наблюдаемость – инструмент, позволяющий выявлять дефекты, проводить непрерывный рефакторинг и предотвращать технический

долг на архитектурном уровне. По оценке Gartner, к 2026 году 70% организаций, применяющие архитектурную наблюдаемость, станут быстрее принимать решения и сохранять конкурентное преимущество [4].

Кроме того, научные исследования показывают, что архитектурные решения как источники долга, так и как средство его снижения, требуют более глубокого понимания и практических инструментов. Это особенно важно, поскольку архитектурный технический долг (ATD) часто превышает по влиянию проблемы, связанные с обычным кодовым техническим долгом.

Таким образом, исследование роли архитектурных решений в управлении и сокращении технического долга является не просто актуальным – оно крайне необходимое для обеспечения устойчивого развития программных систем и рационального использования ИТ-ресурсов в условиях быстро меняющихся технологических и бизнес-требований.

Цель исследования

Цель данного исследования заключается в комплексном анализе роли архитектурных решений в снижении технического долга, выявлении ключевых принципов и подходов, позволяющих минимизировать его накопление на этапе проектирования и сопровождения программных систем, а также в определении эффективных архитектурных практик и инструментов, способствующих поддержанию высокого качества программной архитектуры.

Материалы и методы исследования

В исследовании использованы аналитические обзоры публикаций SEI, DORA, Gartner и CISQ, посвящённых проблемам технического и архитектурного долга. Рассмотрены стандарты ISO/IEC 25010 и ISO/IEC 5055, спецификация CISQ ATDM, а также практики индустриальных инструментов (SonarQube, SQALE). Для анализа практических кейсов применялся сравнительный метод: рассмотрены примеры Amazon Prime Video, Uber (DOMA), Etsy и программы Kessel Run ВВС США. Использованы методы систематизации (классификация видов долга), сравнительный анализ архитектурных стилей (монолит, микросервисы, эволюционная архитектура), а также интерпретация количественных данных (оценки затрат, частота релизов, показатели эффективности DevOps).

Результаты исследования

Термин «технический долг» был введён Уордом Каннингемом в 1990-е годы для объяснения компромиссов в разработке: ускоренный выпуск решений приводит к «процентам» в виде усложнённого сопровождения. Позднее Мартин Фаулер предложил классифицировать долг по намеренности и благоразумию (квадрант), что стало основой современных моделей.

Основные виды технического долга представлены в таблице 1.

Таблица 1

Основные виды технического долга

Вид технического долга	Краткое описание
Кодовый	Дублирование кода, «запахи кода», высокая цикломатическая сложность, нарушение правил стиля и стандарта
Архитектурный	Несоответствие архитектурным принципам (масштабируемость, отказоустойчивость, модульность), наличие «жёстких связей» между компонентами
Дизайнерский	Неправильные зависимости между классами / модулями, нарушение принципов SOLID, избыточная связность
Тестовый	Отсутствие автоматизированных тестов, устаревшие тест-кейсы, низкое покрытие тестами
Документационный	Устаревшая или отсутствующая проектная и пользовательская документация
Долг требований	Неполные, противоречивые или двусмысленные требования, отложенные уточнения
Дефектный	Сознательно оставленные нерешённые дефекты, баги с низким приоритетом
Инфраструктурный	Проблемы в сборочных скриптах, CI/CD конвейерах, зависимостях и окружениях

Полевые исследования SEI фиксируют типичные причины: давление сроков и релизов, недостаточные практики архитектурного анализа, ограниченные тесты / автоматизация, слабая эволюция документации и «короткие пути», принятые для быстрого результата. Результат – рост будущей стоимости изменений, снижение темпа разработки и риски для нефункциональных атрибутов (надёжность, безопасность, производительность) [3].

Для количественной оценки применяются:

– ISO/IEC 5055:2021 – автоматизированные метрики структурных слабостей;

– CISQ ATDM – прогноз затрат на исправление;

– SonarQube – расчёт долга как времени на исправление «code smells»;

– SQALE – модель анализа качества и приоритетов рефакторинга.

Архитектурные решения напрямую влияют на накопление и погашение технического долга, поскольку определяют структуру системы, зависимости между компонентами и возможности для независимого изменения, тестирования и развертывания. Исследования SEI трактуют архитектурный технический долг как краткосрочно удобный подход, создающий контекст, в котором дальнейшие изменения требуют архитектурной «переделки» и обходятся дороже, чем если бы были выполнены своевременно. Это связывает архитектуру с будущей стоимостью изменений и рисками качества [5].

Способность команд быстро и безопасно поставлять изменения коррелирует с архитектурой, позволяющей «слабо связанным» командам автономно проектировать, тестировать и деплоить свой сервис без плотной координации с другими. На практике это выражается в независимых поставках, возможности тестирования без интеграционных «полигонов», контрактном тестировании и эволюционной архитектуре.

К числу ключевых принципов относятся:

1) Модульность и слабая связанность. Чёткое разделение системы на модули с минимальными зависимостями позволяет локализовать изменения и снижает риск распространения дефектов. DORA подчёркивает, что «слабо связанные» архитектуры дают командам возможность автономно разрабатывать и деплоить сервисы без плотной координации с другими командами.

2) Высокая когезия. Каждый модуль или сервис должен решать ограниченный набор задач. Это повышает предсказуемость поведения системы и уменьшает вероятность появления скрытых технических долгов, связанных с избыточной функциональностью.

3) Принцип «открытости / закрытости». Архитектура должна позволять расширять систему без изменения её основных компонентов. Такой подход снижает издержки при масштабировании и модернизации.

4) Явные границы контекста. Использование подхода из DDD (Domain-Driven Design) помогает снизить архитектурный долг за счёт чёткой фиксации правил и зависимостей внутри каждого контекста, избегая «размытия» ответственности между модулями.

5) Эволюционная архитектура. Согласно публикациям ThoughtWorks и AWS, архитектура должна проектироваться так, чтобы изменения могли вноситься постепенно, без «больших взрывов». Применение паттернов типа Strangler Fig или Hexagonal Architecture позволяет уменьшить стоимость изменений и предотвращает накопление долга.

6) Автоматизированное тестирование и инфраструктура как код. Интеграция принципов CI/CD и DevOps в архитектуру (например, через контрактное тестирование и автоматизацию развертываний) снижает риск накопления инфраструктурного и тестового долга.

На рисунке ниже представлена контейнерная диаграмма (нотация C4), иллюстрирующая архитектуру системы доставки. Показаны основные компоненты: веб-приложение, мобильные приложения (iOS, Android, Admin), backend-сервис, а также интеграции с внешними системами доставки («Деловые Линии», «СДЭК», «Возовоз»). Схема демонстрирует взаимодействие между клиентами, администраторами и ключевыми сервисами через API и протоколы HTTPS/JSON.

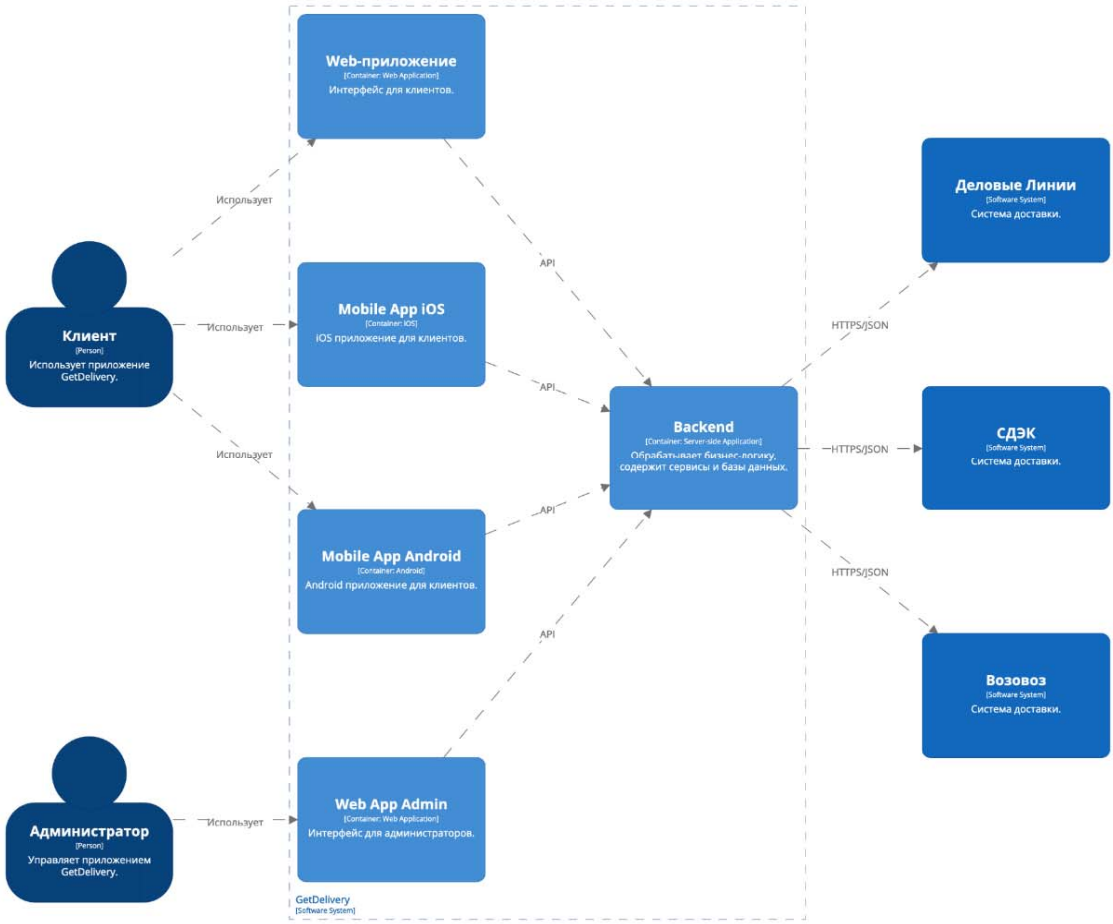

Рис. 1 Схема архитектуры системы доставки (нотация C4) [1]

Примеры успешных практик снижения технического долга за счёт архитектурных решений представлены в таблице 2.

Внедрение архитектурных решений как инструмента управления техническим долгом сопровождается целым рядом проблем и вызовов, подтверждённых как в исследованиях, так и в практическом опыте компаний. Несмотря на очевидные преимущества системной архитектуры — снижение издержек сопровождения, повышение гибкости и скорости разработки, — на пути их реализации возникают организационные, технические и экономические барьеры.

Одним из ключевых вызовов является необходимость сохранять баланс между скоростью вывода продукта на рынок и качеством архитектурных решений. Согласно отчётам DORA и Gartner, многие организации склонны жертвовать архитектурным качеством ради ускорения time-to-market. Это приводит к появлению архитектурного технического долга, который в дальнейшем замедляет разработку и требует больших ресурсов на исправление. Проблема особенно актуальна в условиях высокой конкуренции, когда стратегические архитектурные инициативы часто откладываются в пользу краткосрочных фич.

Таблица 2

Примеры успешных практик снижения технического долга за счёт архитектурных решений

Организация / год	Контекст долга / проблема	Принятое архитектурное решение	Результат (точные данные)
Amazon Prime Video (2023)	Дорогая и сложная в масштабировании распределённая обработка качества видео (VQA)	Консолидация распределённых компонентов в «монолит» на Amazon EC2/ECS; отказ от промежуточного S3 и избыточной оркестрации	Снижение инфраструктурных затрат более чем на 90%; повышение масштабируемости («тысячи потоков») [10]
Uber (DOMA) (2020)	Рост сложности при ~2 200 микросервисах, трудная навигация зависимостей, «сетевой монолит»	Domain-Oriented Microservice Architecture: группировка сервисов в ~70 доменов, слои зависимостей, «gateway» как единая точка входа, расширяемость	Снижение затрат поддержки платформ «на порядок»; сокращение времени приоритизации/ интеграции фичи с 3 дней до 3 часов; сокращение времени онбординга на платформы на 25-50% [8]
Etsy (2014)	Высокая стоимость релизов и «релизный долг» в монолите	Архитектура и практики для CD: один клик деплоя, флаги фич, обязательный мониторинг, «staging≈prod»	>50 деплоев в день при контролируемом риске; культура пост-мортемов и наблюдаемости [7]
U.S. Air Force – Kessel Run (2019)	Экстремально долгие циклы поставки ПО, бюрократия, легаси-процессы	DevSecOps + облачная web-архитектура, event-driven/ микросервисы, непрерывная ATO на защищённых сетях	Среднее время от идеи до эксплуатации ~4,5 мес.; сокращение lead time с 5 лет до 3,5 дней; частота прод-выпусков ~42 в месяц; «continuous ATO» → вывод на защищённую сеть <1 часа [9]
Исследование DORA (2019)	Связь архитектурной связности с показателями эффективности	Рекомендация на слабую связанность и независимые деплои как ключевую способность	«Элита» по сравнению с «низкими»: 208× чаще деплой, 106× быстрее lead time, 2 604× быстрее восстановление, 7× ниже доля неудачных изменений [6]

Исследования SEI и практика крупных ИТ-компаний показывают, что сопротивление изменениям со стороны команд и менеджмента является серьёзным препятствием. Архитектурные решения часто воспринимаются как ограничение свободы действий разработчиков. Недостаток компетенций у инженеров и архитекторов по современным архитектурным стилям (микросервисы, событийно-ориентированные системы, эволюционная архитектура) также ведёт к ошибкам при реализации и росту скрытого долга. Кроме того, отсутствие культуры документирования решений (например, через ADR) усугубляет проблему, приводя к утрате контекста и дублированию ошибок.

Несмотря на наличие стандартов ISO/IEC 5055 и метрик ATDM, оценка архитектурного долга остаётся сложной задачей. Методы количественной оценки часто ограничиваются кодовым уровнем (например, SonarQube), тогда как архитектурные зависимости и организационные аспекты плохо поддаются автоматическому анализу. Это приводит к «слепым зонам», когда значительная часть архитектурного долга накапливается незаметно.

Инвестиции в архитектурные трансформации требуют значительных затрат. Например, миграция с монолита на микросервисную архитектуру сопровождается увеличением расходов на инфраструктуру, обучение персонала и организацию DevOps-процессов. Без чёткой стратегии возврата инвестиций

компании рискуют остаться с двойной нагрузкой – поддержкой легаси и развитием новой архитектуры.

Архитектурный дрейф – ситуация, когда фактическая реализация постепенно отклоняется от проектных решений. Он широко описан в публикациях SEI. Он возникает из-за несогласованности изменений, отсутствия централизованного контроля и устаревшей документации. В долгосрочной перспективе дрейф превращается в скрытый архитектурный долг, усложняющий дальнейшее развитие системы.

Для того чтобы теоретические положения приобрели прикладное значение, необходимо сформулировать конкретные рекомендации для разработчиков и архитекторов, которые помогут снизить риски накопления технического долга и обеспечат устойчивое развитие программных систем.

Ниже представлены практические рекомендации, ориентированные на инженерные команды и архитекторов:

– использовать ADR (Architecture Decision Records) для фиксации решений и снижения скрытого архитектурного долга;

– внедрять эволюционную архитектуру (Strangler Fig, Hexagonal) для постепенной модернизации систем без «больших взрывов»;

– применять метрики и стандарты (ISO/IEC 5055, ATDM, SonarQube/SQALE) для регулярной оценки архитектурного долга;

– обеспечивать слабую связанность и высокую когезию модулей, минимизировать межкомандные зависимости;

– интегрировать архитектурные практики в DevOps/CI/CD, включая контрактное тестирование и инфраструктуру как код.

Перспективы будущих исследований связаны прежде всего с развитием инструментов автоматизированного выявления архитектурного долга. Несмотря на наличие стандартов и метрик, таких как ISO/IEC 5055 и ATDM, большая часть архитектурных проблем остаётся за пределами формализованного анализа. Создание методов, позволяющих в реальном времени фиксировать признаки архитектурного дрейфа и прогнозировать их последствия, станет важным направлением работы.

Отдельного внимания заслуживает использование искусственного интеллекта и машинного обучения для анализа архитектурных моделей и больших массивов инженерных данных. Применение ИИ может повысить точность оценки архитектурного долга, помочь в приоритизации мер по его снижению и даже предлагать альтернативные архитектурные решения.

Кроме того, перспективным направлением является исследование связи архитектурных метрик с бизнес-показателями: скоростью вывода продукта на рынок, стоимостью сопровождения и надёжностью систем. Это позволит компаниям принимать архитектурные решения не только с технической, но и с экономической точки зрения, превращая управление техническим долгом в стратегический инструмент развития.

Выводы

Таким образом, архитектурные решения играют ключевую роль в управлении техническим долгом, определяя способность системы к масштабированию, сопровождению и устойчивому развитию. Практика ведущих

компаний показывает, что применение принципов модульности, слабой связанности и эволюционной архитектуры существенно снижает стоимость изменений и ускоряет поставку программных продуктов. Вместе с тем внедрение архитектурных подходов сопровождается вызовами: балансом между скоростью и качеством, сопротивлением изменениям, недостатком компетенций и рисками архитектурного дрейфа. Для эффективного снижения долга требуется интеграция архитектурных практик в процессы DevOps/CI/CD, использование формализованных инструментов оценки и развитие культуры документирования решений.

Перспективы дальнейших исследований связаны с автоматизацией выявления архитектурного долга, применением ИИ для анализа архитектурных моделей и установлением связи архитектурных метрик с бизнес-показателями, что позволит сделать управление техническим долгом стратегическим инструментом развития компаний.

Список использованной литературы:

1. Нотация моделирования архитектуры C4 – примеры диаграмм и инструменты / Хабр [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/778726/.

2. Что такое архитектурный технический долг и как его исправить [Электронный ресурс]. – Режим доступа: https://www.itweek.ru/management/article/detail.php?ID=229411.

3. A Field Study of Technical Debt [Электронный ресурс]. – Режим доступа: https://www.sei.cmu.edu/blog/a-field-study-of-technical-debt/.

4. Architectural Observability Critical to Managing Tech Debt [Электронный ресурс]. – Режим доступа: https://vfunction.com/blog/why-mastering-architectural-observability-is-pivotal-to-managing-technical-debt/.

5. Architectural Technical Debt Library [Электронный ресурс]. – Режим доступа: https://www.sei.cmu.edu/library/architectural-technical-debt-library/.

6. DORA's 2019 Report: 'DevOps Has Crossed the Chasm' | New Relic [Электронный ресурс]. – Режим доступа: https://newrelic.com/blog/best-practices/dora-accelerate-state-of-devops-2019.

7. How Etsy Deploys More Than 50 Times a Day – InfoQ [Электронный ресурс]. – Режим доступа: https://www.infoq.com/news/2014/03/etsy-deploy-50-times-a-day/.

8. Introducing Domain-Oriented Microservice Architecture | Uber Blog [Электронный ресурс]. – Режим доступа: https://www.uber.com/en-LT/blog/microservice-architecture/.

9. Kessel run: an analysis of the air force’s internal software development organization [Электронный ресурс]. – Режим доступа: https://calhoun.nps.edu/server/api/core/bitstreams/564aa3fd-d576-4aea-a3de-232d82a296b2/content.

10. Scaling up the Prime Video audio/video monitoring service and reducing costs by 90% [Электронный ресурс]. – Режим доступа: https://www.wudsn.com/productions/www/site/news/2023/2023-05-08-microservices-01.pdf.