ORES
Научно-практический электронный журнал
Оригинальные исследования (ОРИС), *ISSN 2222-0402*
www.ores.su
original-research.ru

УДК 004.03

ДЕКОМПОЗИЦИЯ СЛОЖНЫХ СИСТЕМ: СРАВНИТЕЛЬНЫЙ АНАЛИЗ КОМПОНЕНТНЫХ И МИКРОСЕРВИСНЫХ ПОДХОДОВ К МАСШТАБИРОВАНИЮ[1]

Абраш Елизавета Николаевна,
МИРЭА – Российский технологический университет (РТУ МИРЭА),
abrash.e.n@edu.mirea.ru

Аннотация

В статье рассматривается проблематика технической задолженности в микросервисных архитектурах (MSA). Основное внимание уделяется уникальным формам технической задолженности (TD), таким как дублирование функциональности между сервисами и размывание их границ. Обсуждаются проблемы, возникающие из-за отсутствия чёткой архитектурной дисциплины и слабой декомпозиции сервисов. Для анализа TD рассматриваются подход «цель – вопрос – метрика» и инструмент Sonar с индексами SQALE. В результате исследования выявлено, что технический долг остаётся актуальной проблемой в разработке программного обеспечения, особенно с точки зрения MSA. На основании анализа предлагается компонентный подход для минимизации дублирования функциональности и упрощения поддержки.

Ключевые слова: микросервисные архитектуры, техническая задолженность, дублирование кода, распределённые системы, компонентный подход.

DECOMPOSING COMPLEX SYSTEMS: A COMPARATIVE ANALYSIS OF COMPONENT-BASED AND MICROSERVICE-BASED APPROACHES TO SCALING

Abrash Elizabeta Nikolaevna,
MIREA – Russian University of Technology (RTU MIREA),
abrash.e.n@edu.mirea.ru

ABSTRACT

The article discusses the issue of technical debt in microservice architectures (MSA). The focus is on unique forms of technical debt, such as duplication of functionality between services

[1] Коваленко Михаил Андреевич,
Научный руководитель, старший преподаватель МИРЭА – Российский технологический университет (РТУ МИРЭА).
Kovalenko Mikhail Andreevich,
Scientific supervisor, senior lecturer
MIREA – Russian University of Technology (RTU MIREA).

and blurring their boundaries. The problems arising from the lack of a clear architectural discipline and weak decomposition of services are discussed. For the analysis of technical debt, the goal-question-metric approach and the Sonar tool with SQALE indexes are considered. As a result of the study, it was revealed that technical debt remains an urgent problem in software development, especially from the point of view of MSA. Based on the analysis, a component-based approach is proposed to minimize duplication of functionality and simplify support.

Keywords: microservice architectures, technical debt, code duplication, component programming.

Проблематика и актуальность

Переход от монолитных систем к микросервисным архитектурам (MSA) был продиктован необходимостью преодоления ограничений, связанных с тесной связанностью компонентов, сложностью развёртывания и низкой устойчивостью к изменениям. Несмотря на декларируемые преимущества, практика показывает, что MSA не являются полностью эффективным решением – их эволюция порождает специфические проблемы, многие из которых остаются недостаточно изученными.

Технический долг (Technical Debt, TD) – явление, при котором команды разработчиков принимают не самое эффективное решение, что влечёт за собой дополнительные траты на разработку [1]. Технический долг сложно оценить в количественных показателях, при этом его влияние может привести к сбоям в программном обеспечении, неудовлетворённости клиентов, сложностям в поддержке инфраструктуры и дополнительным затратам.

В микросервисах техническая задолженность проявляется в уникальных формах, связанных с распределённой природой системы. Одной из таких форм является дублирование функциональности между сервисами, возникающее из-за неоптимального разделения обязанностей (Bounded Context) [4]. Как отмечают K. Maggi и другие авторы [5], данный конфликт часто происходит, когда разработчики не следуют принципам Domain-Driven Design (DDD), что приводит к нарушению границ сервисов и увеличению сложности поддержки. Такое дублирование усложняет внесение изменений, так как одна и та же логика может быть разбросана по нескольким сервисам [6].

Ещё одной проблемой является размывание границ сервисов при частых изменениях требований, что может привести к появлению так называемых «анти-микросервисов» или распределённых монолитов (distributed monoliths). Как подчёркивается в исследовании [6], отсутствие чёткой архитектурной дисциплины и слабая декомпозиция сервисов способствуют тому, что микросервисы начинают тесно связываться между собой, теряя преимущества независимости. Это увеличивает нагрузку на интеграционные механизмы и усложняет эволюцию системы [7].

Кроме того, в микросервисных архитектурах часто наблюдается накопление так называемых клонов кода, которые маскируются под «независимые реализации», но на практике увеличивают стоимость изменений. Такие клоны могут возникать из-за стремления к изоляции сервисов, но в долгосрочной перспективе решение приводит к дублированию логики и усложнению процесса синхронизации изменений. Это особенно актуально в крупных системах, где разные команды разрабатывают схожие функции, не зная о существующих решениях [8].

Текущие новые и нетипичные формы технической задолженности требуют особого внимания при проектировании и развитии микросервисных систем, так как их

игнорирование может привести к значительному росту затрат на поддержку и модернизацию.

Обзор литературы и методов

В статье «Эволюция технического долга кода в архитектурах микросервисов» [5] для анализа технического долга и сбора данных авторы использовали подход «цель – вопрос – метрика». Авторы отобрали репозитории с открытым исходным кодом на GitHub, соответствующие определённым критериям.

Для отбора репозиториев в исследовании [5] были установлены следующие параметры: основной язык программирования (Java, Python, Go, C#, TypeScript или JavaScript), количество «звёзд» на GitHub, наличие файла docker-compose, промышленное применение.

После извлечения данных из выбранных репозиториев проводилась ручная проверка описания проекта и README-файла для подтверждения качества данных. В качестве инструментов для анализа использовались Sonar и индексы SQALE.

Описанный подход позволил авторам статьи систематически отобрать релевантные репозитории и провести качественный анализ технического долга в микросервисных инфраструктурах.

Результаты исследования показали [5], что технический долг и сейчас остаётся актуальной проблемой в разработке программного обеспечения, в частности при разработке микросервисов. Для успешного управления TD необходимо учитывать его эволюцию, периодичность и связь с архитектурными событиями.

В программных проектах доля клонированного кода варьируется от 7% до 59%. В ходе исследования «Эмпирическое исследование клонов кода: плотность, энтропия и закономерности» [9] выявлено, что плотность клонирования в большинстве проектов колеблется от 0,05 до 0,3. Высокая плотность клонирования связана с наличием нескольких блоков, созданных на основе одного и того же модуля. Также в исследовании [9] было установлено, что энтропия клонирования файлов превышает 0,5 более чем в 75% проектов, в то время как энтропия клонирования пакетов составляет менее 0,5 в более чем 75% проектов. Полученная статистика указывает на то, что клоны кода разбросаны по файлам, но сконцентрированы в определённых пакетах.

Для обнаружения клонов был использован инструмент SAGA. Этот инструмент позволяет проводить крупномасштабные эмпирические исследования и анализировать большие объёмы кода. Для визуализации групп клонов была разработана модель на основе метода древовидной карты.

Хотя инструменты вроде SonarQube и SAGA эффективны для выявления TD и клонов кода, они не учитывают контекст распределённых систем. Например, SonarQube анализирует микросервисы изолированно, игнорируя межсервисные зависимости [6]. Данная особенность затрудняет обнаружение таких проблем, как распределённые монолиты, где высокая связанность между сервисами маскируется под независимость. Кроме того, метрики плотности клонирования (Clone Density) не всегда отражают реальную стоимость поддержки: дублирование в 5% кода может быть критичным, если затрагивает межсервисные API [8].

Авторы исследования «К проблеме сравнения методов детектирования клонов исходного кода» [2] приводят следующие методы обнаружения клонов: текстовый и лексический подходы, алгоритмы на основе деревьев и графов. Особое внимание стоит уделить подходу детектирования клонов кода на основе графов; решение предлагается в виде построения графа зависимостей программы (PDG) во время компиляции проекта и поиска максимально схожих подграфов [3]. Применение алгоритма поможет учесть характеристику распределённой природы микросервисов, однако, анализ проблемы

учитывает традиционные методы разработки программного обеспечения с помощью промежуточного представления LLVM для низкоуровневого кода. Использование такого метода требует дальнейших исследований и адаптации к уникальным проблемам эволюции кода в MSA.

Несмотря на ранее появление первых исследований по обнаружению клонов кода, всё еще остается множество нерешённых проблем и задач [2]. В отечественной научной литературе отсутствуют исследования по созданию универсального алгоритма детектирования, учитывающего высокоуровневые абстракции, несколько языков программирования и взаимосвязи распределённых структур.

В статье «Генеративность и парадокс стабильности и гибкости в архитектуре платформы» [10] на примере Oracle Cloud Platform авторы представляют ключевые технологии, способные генерировать новые продукты и услуги без активного участия их создателя. Такие технологии играют важную роль в современной экономике и бизнесе, способствуя инновациям и росту за счёт реорганизации ИТ-инфраструктуры вокруг гибких и независимых компонентов. Платформы также централизуют сетевые эффекты, возникающие благодаря взаимодействию различных участников экосистемы.

Однако, платформы сталкиваются с парадоксом стабильности и гибкости. Стабильность платформы подразумевает ограничение негативных изменений, что обеспечивает устойчивость и надежность системы. Гибкость, напротив, позволяет платформе адаптироваться к новым условиям и внедрять продуктивные изменения. Парадокс стабильности и гибкости заключается в том, что для реализации генеративности платформа должна одновременно обеспечивать оба этих аспекта.

Стабильность требует контроля и автономии, что позволяет платформе поддерживать высокое качество и надежность. Гибкость, в свою очередь, позволяет избегать препятствий и максимизировать охват рынка. Баланс между этими двумя аспектами является критически важным для реализации способности создавать новые продукты и услуги. Для достижения равновесия между этими аспектами необходима многоуровневая модульная архитектура, где элементы платформы являются функционально различимыми и слабосвязанными.

Современные исследования показывают [11; 12], что выбор между монолитной и микросервисной архитектурой представляет собой сложный компромисс, зависящий от множества факторов. Монолитные системы остаются актуальными в проектах с низкой частотой изменений бизнес-требований, жёсткими требованиями к производительности межкомпонентного взаимодействия и ограниченными ресурсами для поддержки распределённой инфраструктуры. Однако, компонентный подход, занимающий промежуточное положение в эволюции архитектур, обладает рядом преимуществ, таких как более высокая поддерживаемость, меньшая сложность развёртывания и гибкость архитектуры, позволяющая постепенную трансформацию через выделение модулей.

Анализ успешных кейсов миграции позволил выделить трёхфазную модель трансформации архитектуры [12; 13]. На первом этапе, фазе декомпозиции, применяются принципы Domain-Driven Design [12] для выделения границ разделения обязанностей. Метрика успешности – снижение связности между модулями не менее чем на 25%. На втором этапе, фазе инструментирования, внедряется API-First подход с формальными контрактами и организация взаимодействия через шины событий. Критерий готовности – покрытие интеграционных тестов на уровне не менее 90%. На третьем этапе, фазе физического разделения, компоненты постепенно выделяются в автономные сервисы. Ключевой показатель эффективности (KPI) – сокращение метрики средней продолжительности ремонта системы после сбоя или Mean time to repair (MTTR) на не менее чем 40%.

Исследования [5; 11; 12; 13] демонстрируют, что компонентная архитектура остаётся оптимальным выбором для организаций на начальных этапах цифровой трансформации, обеспечивая баланс между гибкостью и сложностью. Проведённый анализ подтверждает, что эволюционный путь системы и, в том числе, кода через компонентную архитектуру является научно обоснованной стратегией миграции к микросервисной инфраструктуре. Перспективные направления будущих исследований включают разработку количественных моделей для оценки готовности к переходу между архитектурными стилями, создание специализированных метрик для оценки эффективности low-code решений в аспектах MSA и исследование гибридных архитектурных паттернов, сочетающих преимущества различных подходов [5]. Особое внимание следует уделить изучению когнитивной нагрузки на команды при различных архитектурных подходах, что может стать отдельной темой для исследования.

Заключение

Для эффективного управления клонированием кода необходимо разработать стратегии, основанные на выявленных шаблонах и причинах дублирования. Такой подход позволит минимизировать избыточность кода и улучшить его обслуживание. Визуализация клонирования кода с помощью метода древовидной карты помогает разработчикам идентифицировать целевые клоны и выявлять шаблоны клонирования. Результаты исследования могут быть использованы для оптимизации процессов разработки и поддержки кода в программных проектах.

Также важно учитывать, что проекты с большим количеством звёзд на GitHub имеют меньшую энтропию клонирования пакетов, что может свидетельствовать о более высоком качестве кода в таких проектах. Это подчёркивает необходимость внедрения практик управления клонами в процесс разработки для повышения критериев качества программного обеспечения.

Следует также обратить внимание на то, что в научной сфере существует недостаток исследований, посвящённых архитектурным компонентам платформ. Большинство исследований фокусируются на управлении платформами, игнорируя особенности их архитектурных моделей. Недостаточное понимание архитектуры платформ приводит к проблемам в их трансформации и адаптации к новым условиям. Кроме того, в отечественной литературе отсутствуют исследования по техническому долгу, учитывающие специфику микросервисной архитектуры.

Основываясь на исследованиях других авторов, можно прийти к выводу, что MSA создают нестандартные вопросы и сложности в процессах управления эволюцией кода, где традиционные подходы к рефакторингу и контролю качества часто оказываются неэффективными. Накопление TD и клонирование кода в таких системах требует разработки новых специализированных метрик, учитывающих распределённую природу сервисов и динамику их взаимодействия. Перспективным представляется направление, объединяющее анализ клонов (Clone Density/Entropy) с архитектурными паттернами (например, Saga Pattern для управления согласованностью), что может стать темой для отдельного и дальнейшего исследования.

Список литературы:

1. Явление технического долга в разработке программного обеспечения и способы борьбы с ним / Поляков Н.Н., Щелкунова М.Е. // Наука, инновации и технологии: от идей к внедрению, Комсомольск-на-Амуре, 16-17 ноября 2023. Комсомольский-на-Амуре гос. университет, 2023. С. 535-537.

2. К проблеме сравнения методов детектирования клонов исходного кода / Д. А. Смутин, Г. А. Якимов, Д. В. Литовкин // Инновационные технологии в обучении и производстве, Волгоград, 19–20 ноября 2024. Том 2. Волгоградский гос. тех. университет, 2024. С. 97-101.
3. Масштабируемый и точный поиск клонов кода / С. Саргсян, Ш. Курмангалеев, А. Белеванцев, А. Аветисян // Программирование. 2015. № 6. С. 9-17.
4. Villa A., Ocharán-Hernández J. O., Pérez-Arriaga J. C., Limón X. A Systematic Mapping Study on Technical Debt in Microservices // 10th International Conference in Software Engineering Research and Innovation (CONISOFT). 2022. pp. 182-191, DOI: 10.1109/CONISOFT55708.2022.00032
5. Maggi K., Verdecchia R., Scommegna L., Vicario E. Evolution of code technical debt in microservices architectures // The Journal of Systems & Software. 2025. Vol. 222. 112301, DOI: 10.1016/j.jss.2024.112301
6. Verdecchia R., Maggi K., Scommegna L., Vicario E. Technical debt in microservices: A mixed-method case study // ECSA 2023. 2024. pp. 217-236, DOI: 10.1007/978-3-031-66326-0_14
7. Bogner J., Fritzsch J., Wagner S., Zimmermann A. Assuring the Evolvability of Microservices: Insights into Industry Practices and Challenges // IEEE International Conference on Software Maintenance and Evolution (ICSME). 2019. pp. 546-556, DOI: 10.1109/ICSME.2019.00089
8. B. van Bladel, Murgia A., Demeyer S. An empirical study of clone density evolution and developer cloning tendency // IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER). 2017. pp. 551-552, DOI: 10.1109/SANER.2017.7884672
9. Hu B., Yu D., Wu Y., Hu T., Cai Y. An empirical study of code clones: Density, entropy, and patterns // Science of Computer Programming. 2025. Vol. 242. 103259, DOI: 10.1016/j.scico.2024.103259
10. Sun R., Gregor S., Fielt E. Generativity and the paradox of stability and flexibility in a platform architecture: A case of the Oracle Cloud Platform. // Information & Management. 2021. Vol. 58. I. 8. 103548, DOI: 10.1016/j.im.2021.103548
11. Ana Martínez Saucedo, Guillermo Rodríguez, Fabio Gomes Rocha, Rodrigo Pereira dos Santos Migration of monolithic systems to microservices: A systematic mapping study // Information and Software Technology. 2025. Vol. 177, 107590, DOI: 10.1016/j.infsof.2024.107590
12. Mohottige T. I., Polyvyanyy A., Fidge C., Buyya R., Barros A. Reengineering software systems into microservices: State-of-the-art and future directions // Information and Software Technology. 2025. Vol. 183. 107732, DOI: 10.1016/j.infsof.2025.107732
13. Hüseyin Ünlü, Dhia Eddine Kennouche, Görkem Kılınç Soylu, Onur Demirörs Microservice-based projects in agile world: A structured interview // 2024. Information and Software Technology. 2024. Vol. 165. 107334, DOI: 10.1016/j.infsof.2023.107334