

ДЕКОМПОЗИЦИЯ МОНОЛИТНОГО ПРИЛОЖЕНИЯ НА МИКРОСЕРВИСЫ НА ОСНОВЕ АНАЛИЗА МЕТРИК СВЯЗНОСТИ

В.Э. Буздин, Е.А. Аникеев

ФГБОУ ВО «Воронежский государственный лесотехнический университет
имени Г.Ф. Морозова»

Аннотация. Рассмотрен подход к выделению границ микросервисов при разбиении монолитного приложения по анализу графа зависимостей между модулями. Связи кода представлены матрицей вызовов, по которой рассчитаны метрики связанности и сцепления, а группировка модулей выполнена по критерию модульности. На учебном проекте показано снижение средней межмодульной связности и доли кросс-модульных вызовов после декомпозиции.

Ключевые слова: микросервисы, декомпозиция, монолит, граф зависимостей, связанность, сцепление, метрики, кластеризация, архитектура программного обеспечения.

DECOMPOSITION OF A MONOLITHIC APPLICATION INTO MICROSERVICES BASED ON COUPLING METRICS ANALYSIS

V.E. Buzdin, E.A. Anikeev

Voronezh State University of Forestry and Technologies named after G.F. Morozov

Abstract. An approach to defining microservice boundaries when splitting a monolithic application based on the analysis of the module dependency graph is considered. Code links are represented by a call matrix, from which coupling and cohesion metrics are calculated, and modules are grouped by the modularity criterion. Using an educational project, a reduction of the average inter-module coupling and the share of cross-module calls after decomposition is shown.

Keywords: microservices, decomposition, monolith, dependency graph, coupling, cohesion, metrics, clustering, software architecture.

Введение

Разработка корпоративных приложений долгое время велась в монолитной архитектуре: весь функционал собирается в один развёртываемый артефакт с общей базой данных. На старте проекта это удобно, но по мере роста кодовой базы проявляются известные ограничения: сборка и тестирование занимают всё больше времени, цена изменения растёт, отдельные части системы нельзя масштабировать независимо, а локальная ошибка способна остановить приложение целиком. Переход к микросервисной архитектуре снимает часть этих ограничений за счёт того, что сервисы развиваются и развёртываются по отдельности.

Однако наиболее трудным решением при таком переходе остаётся выбор границ сервисов. Неудачная декомпозиция приводит к «распределённому монолиту»: сервисы формально разделены, но связаны настолько плотно, что почти любое изменение затрагивает сразу несколько из них. В результате система получает все издержки распределённости, но не получает главного преимущества — независимости разработки и развёртывания.

На практике границы часто проводят по интуиции или по сложившейся структуре команд. Такой выбор трудно обосновать и ещё труднее проверить. Между тем связи в коде объективно фиксируются: это вызовы между модулями, обращения к общим данным, совместное использование типов. Это позволяет поставить задачу декомпозиции как задачу системного анализа [3] — представить приложение графом зависимостей и искать такое разбиение, при котором сильно связанные элементы остаются вместе, а слабо связанные оказываются в разных сервисах. Близкий по духу приём — архитектурный анализ исходного кода для оценки качества проекта — рассматривается в работе [4].

Цель работы — предложить и проверить процедуру выделения границ микросервисов по метрикам связности, рассчитанным на графе зависимостей модулей. В отличие от разбиения «по чутью», такая процедура даёт воспроизводимый результат и числовое обоснование, которое можно показать на ревью и в нормоконтроле.

Исходные данные и постановка задачи

В качестве объекта рассмотрено учебное веб-приложение интернет-магазина запасных частей, написанное в монолитной архитектуре. Кодовая база включает 34 модуля (пакета), сгруппированных вокруг шести предметных (бизнес-) областей [1]: оформление заказов, каталог товаров, оплата, склад, профиль пользователя и отчётность. Все модули работают с единой базой данных и собираются в один артефакт.

Зависимости между модулями извлекались статически — из импортов и вызовов в исходном коде, а также из обращений к таблицам базы данных. Каждой паре модулей сопоставлялось число направленных связей: сколько раз код одного модуля обращается к другому. Полученные данные представлены ориентированным графом зависимостей $G = (V, E)$, где вершины V — модули, а вес ребра отражает интенсивность обращений. Этот граф и стал исходным материалом для анализа.

Задача формулируется так: разбить множество модулей V на непересекающиеся группы (будущие сервисы) так, чтобы связи внутри групп были как можно плотнее, а между группами — как можно реже. Дополнительно учитывались два практических ограничения: модули, работающие с одними и теми же таблицами, желательно держать в одном сервисе (иначе возникает распределённая транзакция), а размер сервиса не должен быть слишком большим или слишком малым.

Метрики связности и модель зависимостей

Для числовой оценки использовались две классические характеристики качества кода из объектно-ориентированного проектирования [2]. Связанность модуля (coupling between objects, CBO) — число других модулей, с которыми данный модуль связан напрямую; чем она выше, тем сильнее модуль зависит от окружения. Сцепление внутри модуля (косвенно через метрику LCOM) показывает, насколько элементы модуля решают одну задачу; высокое сцепление желательно, низкое говорит о том, что модуль стоит разделить.

Качество разбиения графа на группы оценивалось по модульности Q — показателю, широко применяемому при анализе сетей. Он сравнивает долю

связей, попавших внутрь групп, с той долей, которая получилась бы при случайном распределении связей той же интенсивности:

$$Q = (1 / 2m) \cdot \sum_{ij} [A_{ij} - (k_i \cdot k_j) / 2m] \cdot \delta(c_i, c_j) \quad (1)$$

где A_{ij} — вес связи между модулями i и j ; k_i — суммарный вес связей модуля i ; m — суммарный вес всех связей графа; c_i — номер группы, в которую отнесён модуль i ; δ равна единице, если оба модуля в одной группе, и нулю в противном случае. Значение Q растёт, когда плотно связанные модули собираются в одну группу, поэтому максимизация Q по формуле (1) и даёт искомое разбиение на сервисы.

Низкая связанность между сервисами (внешний СВО) и высокое сцепление внутри сервиса — это и есть формальные признаки удачной декомпозиции. Поэтому метрики считались дважды: до разбиения (для всех модулей монолита) и после (для модулей, сгруппированных в сервисы), а сравнение этих значений служило критерием успеха.

Кластеризация модулей и выделение сервисов

Группировка модулей выполнялась жадным агломеративным методом, направленным на рост модульности. Сначала каждый модуль считается отдельной группой. Затем на каждом шаге объединяются две группы, слияние которых даёт наибольший прирост Q ; процесс продолжается, пока объединение увеличивает показатель. Полученное разбиение корректировалось вручную с учётом ограничения по общим таблицам базы данных: если два кандидата в разные сервисы работали с одной таблицей, они объединялись либо для одного из них предусматривался отдельный программный интерфейс доступа к данным.

Слева на рисунке 1 показан исходный монолит: модули плотно переплетены перекрёстными вызовами, граница предметных областей размыта. Справа — результат декомпозиции: пять сервисов с выраженными внутренними связями, взаимодействующих только через интерфейсы (API). Такой вид взаимодействия принципиален: после разбиения сервисы не обращаются к чужим данным напрямую, а запрашивают их у владельца.

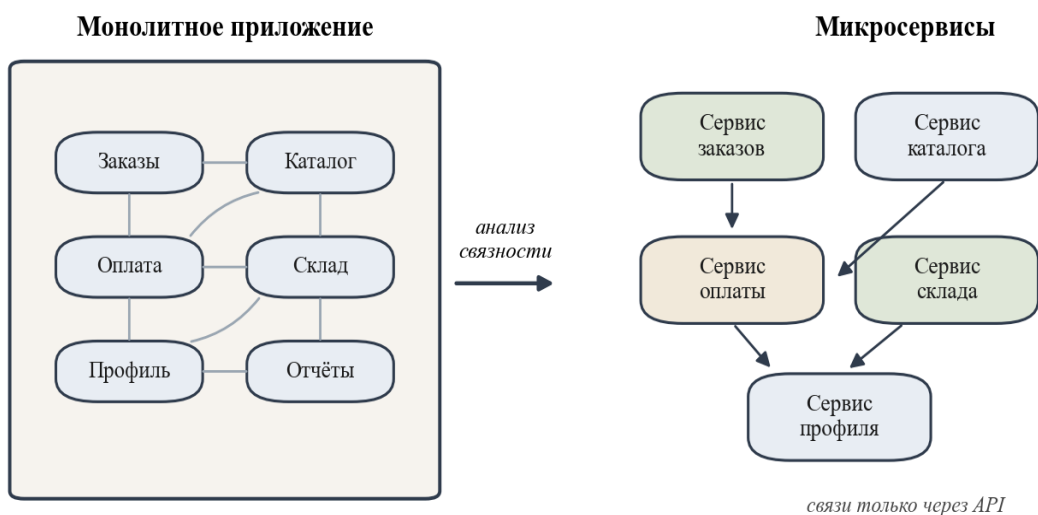


Рисунок 1 – Переход от монолита с перекрёстными связями к набору микросервисов, взаимодействующих через интерфейсы

Результат группировки сведён в таблицу 1. Для каждого выделенного сервиса приведены число вошедших модулей, среднее значение связанности внутри сервиса и число внешних связей с другими сервисами. Видно, что внутренняя связанность заметно превышает внешнюю — это и есть желаемое соотношение для самостоятельного сервиса.

Таблица 1 – Характеристики выделенных сервисов

Сервис	Модулей	Внутренних связей (ср.)	Внешних связей
Заказы	8	5,9	4
Каталог	6	4,7	2
Оплата	5	4,2	3
Склад	7	5,1	2
Профиль	8	4,5	1

Результаты декомпозиции

Сравнение метрик до и после разбиения приведено на рисунке 2. Средняя связанность модуля (СВО) снизилась с 12,4 до 4,1, наибольшая связанность — с 24 до 7, а среднее число зависимостей на модуль — с 6,8 до 2,3. Доля кросс-модульных вызовов, выходящих за границу будущего сервиса, сократилась с 41 % до 12 %. Иными словами, после декомпозиции подавляющее большинство связей замкнулось внутри сервисов, а наружу осталась лишь небольшая, контролируемая часть взаимодействий.

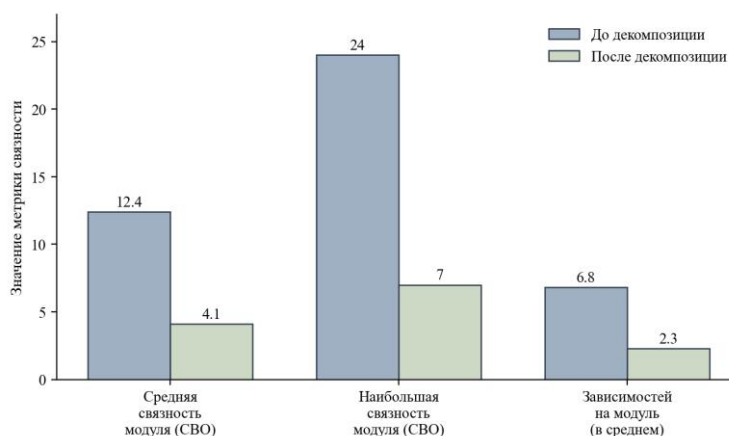


Рисунок 2 – Метрики связанности модулей до и после декомпозиции

Модульность разбиения Q , рассчитанная по формуле (1), составила 0,62, что для графа такого размера соответствует хорошо выраженной структуре групп. Важно, что полученные границы совпали с предметными областями приложения лишь частично: например, модуль формирования отчётов, который интуитивно относили к отдельной области, по связям оказался ближе к сервису заказов и был включён в него.

Дополнительно оценивалась независимость развёртывания. До разбиения изменение в любом из шести направлений требовало пересборки и повторного тестирования всего приложения. После выделения сервисов изменение, не затрагивающее межсервисные интерфейсы, ограничивалось одним сервисом: по выборке из 40 типовых задач сопровождения доля изменений, локализованных в одном сервисе, составила 78 %.

Обсуждение

Предложенная процедура переводит спор о границах сервисов из области вкуса в область измеримых величин. Связи берутся из самого кода, метрики считаются единообразно, а разбиение воспроизводимо. Это удобно и на ревью,

и при нормоконтроле: выбор каждой границы можно обосновать числом, а не общими словами.

У подхода есть и ограничения. Статический анализ не учитывает интенсивность связей во время работы: редкий, но критичный по данным вызов он оценивает так же, как частый, но безобидный. Не охватываются и неявные зависимости — через очереди сообщений, общий кэш или конфигурацию. Поэтому числовое разбиение разумно рассматривать как первое приближение, которое уточняется по эксплуатационным данным и знаниям предметной области. Сопоставление архитектурных вариантов по нескольким критериям, в том числе с привлечением экспертных моделей, обсуждается в работах [5, 6].

Заключение

Показано, что границы микросервисов можно выделять по анализу графа зависимостей, а не «по чутью». Связи модулей представлены матрицей вызовов, по ней рассчитаны метрики связанности и сцепления, а группировка выполнена по критерию модульности. На учебном проекте средняя связанность модуля снизилась втрое, доля кросс-модульных вызовов упала с 41 % до 12 %, а большинство изменений после разбиения стали локализоваться в одном сервисе. В итоге процедура даёт воспроизводимое и числово обоснованное разбиение монолита, пригодное как отправная точка для перехода к микросервисной архитектуре.

Список литературы

1. Никитин, И. В. Сравнение подходов монолитной архитектуры и микросервисной архитектуры при реализации серверной части веб-приложения / И. В. Никитин, Т. Ю. Гриценко // Дневник науки. – 2020. – № 3(39). – С. 22. – EDN TTOGCE.
2. Караханова, А. А. Анализ микросервисной архитектуры, монолитных приложений, архитектуры SOA / А. А. Караханова // Синергия Наук. – 2020. – № 46. – С. 255-262. – EDN BVOVEU.
3. Тараскин, А. А. Экспериментальное сравнение производительности монолитной и акторной архитектур для дашбордов реального времени /

А. А. Тараскин, А. Р. Харисов // Вестник науки. – 2025. – Т. 3, № 12(93). – С. 1413-1422. – EDN ZCMWEQ.

4. Минакова, О. В. Повышение эффективности работы в проектах open source на основе архитектурного анализа (на примере проекта Сахана) / О. В. Минакова, И. В. Поцебнева, П. Ю. Гусев // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 84-92. – DOI 10.12737/2219-0767-2024-17-1-84-92. – EDN NZNSKM.

5. Курипта, О. В. Архитектурное решение проектирования сервисов пространственно-временной навигации в образовательных учреждениях / О. В. Курипта, О. В. Минакова, И. В. Поцебнева // Моделирование систем и процессов. – 2024. – Т. 17, № 1. – С. 65-72. – DOI 10.12737/2219-0767-2024-17-1-65-72. – EDN CNSEZA.

References

1. Nikitin, I. V. Comparison of Monolithic and Microservice Architecture Approaches for Implementing the Server Side of a Web Application / I. V. Nikitin, T. Yu. Gritsenko // Science Diary. – 2020. – No. 3(39). – P. 22. – EDN TTOGCE.

2. Karakhanova, A. A. Analysis of Microservice Architecture, Monolithic Applications, and SOA Architecture / A. A. Karakhanova // Synergy of Sciences. – 2020. – No. 46. – Pp. 255-262. – EDN BVOVEU.

3. Taraskin, A. N. A. N. Experimental performance comparison of monolithic and actor architectures for real-time dashboards / A. N. A. N. Taraskin, A. N. Ph. D. Kharisov // Bulletin of Science. – 2025. - Vol. 3, № 12(93). - St. 1413-1422. – EDN ZCMWEQ.

4. Minakova, O. V. Improving the efficiency of work in open source projects based on architectural analysis (using the example of the Sakhana project) / O. V. Minakova, I. V. Potsebneva, P. Yu. Gusev // Modeling of systems and processes. – 2024. – Vol. 17, No. 1. – pp. 84-92. – DOI 10.12737/2219-0767-2024-17-1-84-92. – EDN NZNSKM.

5. Kuripta, O. V. Architectural solution for designing space-time navigation services in educational institutions / O. V. Kuripta, O. V. Minakova, I. V. Potsebneva // Modeling of systems and processes. – 2024. – Vol. 17, No. 1. – pp. 65-72. – DOI 10.12737/2219-0767-2024-17-1-65-72. – EDN CNSEZA.