Информатика сообществ и формирование социальных сетей

Модульный подход в разработке информационных систем: преимущества и реализация

Д. Рахмани, А.А. Дубровская, А.А. Лобанова

Московский технический университет связи и информатики, г. Москва, Россия

Аннотация. В статье рассматриваются ключевые принципы модульного проектирования, включая инкапсуляцию, повторное использование компонентов и микросервисный подход. Анализируются примеры реализации модульных систем в корпоративном программном обеспечении, веб-приложениях и облачных платформах. Оцениваются преимущества, такие как упрощение сопровождения и тестирования, а также возможные сложности, связанные с межмодульной интеграцией. Применены теоретические методы исследования, такие как анализ существующих практик проектирования, классификация подходов по критериям гибкости и масштабируемости, формализация принципов модульного проектирования, на основе которой можно определить насколько хорошо спроектирована архитектура системы, а также аналогия, на основе которой проведено сравнение микросервисной архитектуры и классического модульного подхода. Результаты исследования включают систематизацию методов интеграции модулей и выявление рисков межмодульной коммуникации.

Ключевые слова: *Модульная архитектура, повторное использование компонентов, микросервисный подход, инкапсуляция, масштабируемость, тестирование модулей, межмодульная интеграция.*

DOI: 10.14357/20790279250306 **EDN:** OTABGG

Введение

Работа освещает важную и современную тему модульного проектирования, рассматривая ее преимущества, реализацию и связь с архитектурными концепциями, такими как микросервисы и SOA (Service-Oriented Architecture). В современных условиях, когда требования к информационным системам постоянно меняются, модульный подход становится основой для гибкости, масштабируемости и упрощения сопровождения.

При рассмотрении модульного подхода в исторической ретроспективе важно выделить процесс эволюции от монолитных систем. Переход на модульную архитектуру стал необходимостью, когда монолитная архитектура начала ограничивать эффективность ИС. Все чаще системы стали сталкиваться с задержками в обновлениях из-за

сложностей в тестировании и внедрении, а также с необходимостью масштабировать отдельные функции приложения независимо друг от друга.

Современные информационные системы становятся все сложнее, требуя высокой гибкости, масштабируемости и простоты поддержки. Главная задача при разработке больших систем — снижение сложности.

Одним из наиболее эффективных способов достижения этих целей является *модульная архитектура*, которая позволяет разбивать систему на независимые, но взаимодействующие компоненты. Это решение не только универсально, но и обеспечивает:

- Гибкость – возможность изменять отдельные части без перестройки всей системы.
- Масштабируемость – независимое масштабирование компонентов.
- Устойчивость – дублирование критически важных модулей повышает отказоустойчивость.

Чем более независимы подсистемы, тем безопаснее разрабатывать каждую из них отдельно, не заботясь обо всей системе.

Модульная архитектура предполагает разбиение системы на автономные блоки (модули) с четкой областью ответственности и минимальными связями. Это упрощает:

- Поддержку – можно обновлять модули независимо.
- Тестирование – изолированная проверка функциональности.
- Масштабирование – нагрузка распределяется между модулями.

Основные свойства модулей:

- Называемость – модуль можно вызывать по имени.
- Инкапсуляция – внутренняя реализация скрыта, взаимодействие только через API.
- Независимость – изменения в одном модуле не ломают другие.
- Малый размер – для упрощения понимания и поддержки.

Эти принципы нашли развитие в SOA (Service-Oriented Architecture) и микросервисах, где модули становятся полностью независимыми сервисами.

1. Основная часть

Технология SOA – это модульный подход к разработке программного обеспечения, основанный на использовании сервисов (служб) со стандартизированными интерфейсами. В основе SOA лежат принципы многократного использования функциональных элементов ИТ, ликвидации дублирования функциональности в программном обеспечении, унификации типовых операционных процессов. Компоненты сервиса могут быть распределены по разным узлам сети. Программные комплексы, разработанные в соответствии с SOA, часто реализуются как набор веб-сервисов, интегрированных при помощи известных стандартных протоколов (SOAP, WSDL и т. п.).

Необходимо отметить следующую особенность технологии SOA: она не только позволяет реализовать адаптивность ИС к изменениям в бизнес-процессах за счет расширения ее функциональности, но и решает проблемы интеграции приложений и данных. SOA можно определить как возможность наращивания прикладного функционала ИС путем создания новых композитных приложений на базе готовых сервисов с использованием унифицированных интеграционных платформ (типа технологии Enterprise Service Bus, реализующей унифицированный механизм взаимодействия приложений). Современным подходом к проектированию адаптивных ИС является также направление облачных технологий, которое позволяет реализовать интеграцию существующих бизнес-приложений с облачными приложениями и сервисами [2, с. 107].

Основные принципы модульного проектирования:

- Независимость модулей: изменения в одном модуле не ломают работу других. Если функциональность модуля больше не нужна, достаточно удалить его папку, и система продолжит работать без этой части функционала.
- Инкапсуляция: все, что относится к модулю (компоненты, стили, утилиты, логику), хранится внутри одной папки. Взаимодействовать с модулем снаружи можно только через Public API, что защищает внутреннюю реализацию от внешних зависимостей.
- Зависимость только от «ядра» (core): если модулям нужны общие сервисы (например, логгер, сетевые запросы, роутинг), они берут их из ядра. Модули не должны напрямую зависеть друг от друга.
- Однонаправленный поток: данные идут сверху вниз: pages ⇒ modules ⇒ components ⇒ UI. Такая структура упрощает понимание, в каком месте обрабатывается бизнес-логика.

Эти идеи находят свое максимальное выражение в микросервисной архитектуре, где модули становятся полностью независимыми сервисами с отдельным развертыванием. Микросервисный подход расширяет принципы модульного проек-

Табл. 1

Различия модульного проектирования и микросервисной архитектуры.

Критерий	Модульное проектирование	Микросервисная архитектура
Границы	Логическое(классы, пакеты)	Физические (отдельные процессы/контейнеры)
Масштаб	Уровень кода	Уровень развертывания
Взаимодействие	Внутрипроцессное (вызовы методов)	Межпроцессное (сеть, API)
Гибкость	Легко изменить, но в рамках одной кодовой базы	Можно развертывать и масштабировать независимо

тирования, применяя их к распределенным системам. Данный подход сохраняет идеи разделения ответственности и слабой связности, но добавляет новые возможности за счет физического разделения компонентов [3].

Различия модульного проектирования и микросервисной архитектуры представлены в табл. 1.

В рамках модульного подхода могут использоваться типовые проектные решения (ТПР), то есть ТПР является одним из способов реализации модульной архитектуры.

Типовое проектное решение – решение, пригодное к многократному использованию (тиражируемое проектное решение). Применение методов типового проектирования имеет свои особенности. Основным условием для использования таких методов является возможность декомпозиции проектируемой ИС на составляющие компоненты (подсистемы, программные модули, комплексы выполняемых задач и тд), для реализации которых можно выбрать типовые проектные решения, существующие на рынке, которые будут настроены на нужды конкретного предприятия. Помимо собственно функциональных (программных, аппаратных) элементов, типовое решение подразумевает наличие необходимой документации, в которой дается детальное описание ТПР (в т.ч. процедур настройки), отвечающее требованиям проектируемой системы [1, с. 55]

По уровню декомпозиции системы можно выделить такие классы ТПР, как:

- элементные ТПР – ТПР по отдельному элементу (задаче, виду обеспечения);
- подсистемные ТПР – ТПР по отдельным подсистемам;
- объектные ТПР – отраслевые ТПР, включающие весь набор подсистем ИС.

2. Преимущества и сложности модульного принципа разработки программ

Модульный принцип разработки программ обладает следующими преимуществами:

- большую программу могут разрабатывать одновременно несколько исполнителей, и это позволяет сократить сроки ее разработки;
- появляется возможность создавать и многократно использовать в дальнейшем библиотеки наиболее употребляемых программ;
- упрощается процедура загрузки больших программ в оперативную память, когда требуется ее сегментация;
- возникает много естественных контрольных точек для наблюдения за осуществлением хода разработки программ, а в последующем – для контроля за ходом исполнения программ;
- обеспечивается более эффективное тестирование программ, проще осуществляются проектирование и последующая отладка.

Преимущества модульного принципа построения программ особенно наглядно проявляются на этапе сопровождения и модификации программных продуктов, позволяя значительно сократить затраты сил и средств на реализацию этого этапа [7].

Однако при применении модульного проектирования, можно столкнуться и с некоторыми сложностями, связанными с межмодульной интеграцией.

Основные проблемы, с которыми можно столкнуться при применении модульного проектирования:

1. Несовместимость интерфейсов: так как модули разрабатываются разными командами (в разное время), то это может привести к нестыковке API, форматов данных или протоколов взаимодействия.
2. Могут возникнуть зависимости между модулями: они усложняют обновление и замену компонентов.
3. Несовместимость версий модулей при обновлении. Модули на разных языках/фреймворках могут требовать дополнительных адаптеров
4. Возникает сложность в тестировании, так как интеграционные тесты требуют развертывания всех зависимых модулей, что увеличивает ресурсы и время.

Для минимизации проблем важно конкретнее определять границы модулей (создавать более гибкие и поддерживаемые системы, ориентируясь на бизнес-логику), иметь более развитую инфраструктуру интеграции, автоматизировать тестирование и развертывание.

Несмотря на возможные проблемы, с которыми можно столкнуться при использовании модульного подхода, он является наиболее масштабируемым и позволяет снизить сложность разработки и внедрения. Вышеперечисленные недостатки могут быть устранены при правильном подходе со стороны команды сопровождения и разработки, а также при использовании единых стандартов как в разработке, так и во внедрении. Таким образом, проблемы не являются останавливающими при выборе определенного подхода к проектированию.

3. Реальные примеры применения модульности в IT-системах

Для практического обоснования значимости модульной разработки необходимо отметить компании, успешно перешедшие на модульную архитектуру [6]**:**

1. Netflix
   - Было: монолитное Java-приложение для видеостриминга.
   - Стало: микросервисная архитектура (сотни сервисов: рекомендации, биллинг, стриминг).
   - Эффект от перехода: масштабируемость, отказоустойчивость, независимое развертывание.
2. Amazon
   - Было: монолит на C++ и Perl.
   - Стало: SOA → микросервисы (AWS, сервисы корзины, рекомендаций).
   - Эффект от перехода: ускорение разработки, уменьшение зависимостей между командами.
3. Uber
   - Было: монолит на Python и Node.js.
   - Стало: микросервисы (геолокация, оплата, поездки).
   - Эффект от перехода: глобальная экспансия, необходимость масштабирования.
4. Spotify
   - Было: монолит с жесткой связностью.
   - Стало: микросервисы + гибридная архитектура (рекомендации, плейлисты).
   - Эффект от перехода: гибкость, быстрый выпуск новых функций.
5. Twitter
   - Было: монолит на Ruby on Rails.
   - Стало: микросервисы на Scala/Java (твиты, лента, уведомления).
   - Эффект от перехода: проблемы с масштабированием при росте пользователей.

4. Формализация принципов с использованием математического аппарата

Сейчас широко используется формализация принципов модульного проектирования с использованием математического аппарата (например, графа), где узлы – это модули, а ребра – зависимости между ними. Это позволяет анализировать связанность системы и минимизацию связей между модулями. Также, математическое описание модульного проектирования формализует принципы для количественной оценки качества архитектуры. Ниже приведены выражения для справки как основные формулы формализации признаков [4]:

1. Графовые модели модульных систем. Теория графов помогает формально описывать и анализировать модульную архитектуру программной системы. Ее можно формализовать с помощью теории графов, где:
   - Вершины (узлы) → модули системы (M_1, M_2, M_3).
   - Ребра → зависимости между модулями.

   Пример графа для системы из трех модулей:

$$M_1 \rightarrow M_2 \rightarrow M_3$$

   Здесь:
   - M_1 зависит от M_2.
   - M_2 зависит от M_3.
   - M_3 является «листом» (не имеет зависимостей).

   Матрица смежности для этого графа:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

   Анализ:
   - Чем плотнее матрица (больше единиц), тем выше связность системы.
   - Идеальная модульность стремится к блочно-диагональной матрице, где ненулевые элементы сгруппированы в небольшие кластеры.
2. Метрики оценки качества модульности [5]. Количественные показатели помогают оценить,

насколько хорошо спроектирована модульная архитектура системы.

Для количественной оценки архитектуры используются:

1. Сцепление (Coupling).
   Мера зависимости между модулями. Формально вычисляется как:
   C = Σ (внешние связи модуля) / Σ (все возможные связи).
   Идеал: $C \to 0$ (минимальные зависимости).
2. Связность (Cohesion).
   Мера внутренней целостности модуля. Вычисляется через:
   L = 1 – (Σ связей между элементами модуля / Σ всех возможных внутренних связей).
   Идеал: L → 0 (высокая внутренняя связанность).
3. Индекс модульности (Q).
   Из теории сложных сетей:

$$Q = \sum \left[e_{ii} - \left(\sum e_{ij} \right)^2 \right],$$ где e_{ij} – доля |связей между модулями i и j

Интерпретация:

- ${}_3Q \in [-0.5 ; 1]$.
- $Q > 0.3 \to$ хорошая модульность.

3. Формальная валидация архитектуры. Методы анализа программной архитектуры с помощью теории графов и математических моделей предназначены для выявления потенциальных проблем (например, узких мест, избыточных зависимостей и оптимизации структуры системы).

Анализ на основе графов:

1. Вычисление центральности модулей:
   - Если один модуль имеет высокую степень центральности, он становится «узким местом».
   - Пример: в графе $M_1 \to M_2 \to M_3$ модуль M_2 — критическая точка отказа.
2. Поиск сильно связанных компонент:
   - Алгоритмы вроде алгоритма Лоя-Унгера выявляют группы модулей, которые стоит объединить.
3. Циклические зависимости:
   - Обнаруживаются через поиск циклов в ориентированном графе (алгоритм Тарьяна).
   - Пример проблемы:
     $M_1 \to M_2 \to M_3 \to M_1$ // Цикл из трех модулей.

Математическая модель оптимальной модульности. Цель: минимизировать функцию:

$$F = \alpha * C + \beta * (1 - L) + \gamma * Q^{-1} \to min,$$

где: α, β, γ — весовые коэффициенты; C — сцепление, L — связность, Q — индекс модульности.

Пример анализа реальной системы.

Дано:

- Модуль Payment зависит от Auth и Database;
- Auth зависит от Database;
- Logging независим.

Граф:

Payment → Auth → Database
Payment → Database
Logging

Матрица смежности:

$$\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Выводы:

1. ***Database*** — «корневой» модуль (много зависимостей).
2. ***Logging*** — идеально изолирован.
3. ***Payment*** имеет высокое сцепление (C=0.5).

Заключение

В заключении отметим, что проведенное исследование подтвердило актуальность модульной архитектуры и данный подход по-прежнему остается фундаментом современной разработки информационных систем. С использованием методов анализа и классификации удалось выявить ключевые направления эволюции модульности: переход от монолитных структур к модульной архитектуре, масштабирование функций приложения независимо друг от друга, интеграция облачных сервисов с сохранением принципов инкапсуляции. Применение метода аналогии микросервисов и модульных систем позволило выявить общие паттерны декомпозиции и критические различия между данными подходами. Применение формализации позволило понять как наиболее эффективно оценить качество архитектуры, а также выявить ее потенциальные проблемы. С развитием технологий принципы модульности эволюционируют, адаптируясь к вызовам облачных платформ, искусственного интеллекта и распределенных архитектур. В перспективе модульный подход будет развиваться в сторону еще большей гибкости и адаптивности.

Литература

1. *Коцюба И.Ю. и др.* Основы проектирования информационных систем // Учебное пособие. Университет ИТМО. 2015.
2. *Рахманов В.И.* К вопросу построения адаптивных информационных систем / В. И. Рахманов // Инновационная наука. 2015. № 10-1. С. 106-108.
3. *Гольчевский Ю.В.* Актуальность использования микросервисов при разработке информационных систем / Ю. В. Гольчевский, А. В. Ермоленко // Вестник Сыктывкарского университета. Серия 1: Математика. Механика. Информатика. 2020. № 2(35). С. 25–36.
4. *Игнацкая И.В.* Представление и анализ архитектуры программных систем на основе графа взаимодействий / И.В. Игнацкая // Труды МАИ. 2010. № 39. 16 с.
5. *Martin R.C.* Agile Software Development: Principles, Patterns, and Practices. 2002.
6. Nigel Pereira finds out why big tech moved from monolithic applications to a superior form of programming in the form of microservices [Электронный ресурс]. URL: https://www.sify.com/digital-transformation/why-amazon-netflix-and-uber-prefermicroservices-over-monoliths/ (дата обращения 09.04.2025).
7. Introduction to Modularity and Interfaces In System Design [Электронный ресурс]. URL: Introduction to Modularity and Interfaces In System Design | GeeksforGeeks (дата обращения 11.04.2025)

Рахмани Джахед. Московский технический университет связи и информатики, г. Москва, Россия. Старший преподаватель. Область научных интересов: информационные технологии, разработка ИС. E-mail: j.rahmani@mtuci.ru.

Дубровская Александра Андреевна. Московский технический университет связи и информатики, г. Москва, Россия. Студентка. Область научных интересов: информационные технологии, разработка ИС, большие данные и аналитика данных. E-mail: sashadubrovskay2005@gmail.com.

Лобанова Анна Алексеевна. Московский технический университет связи и информатики, г. Москва, Россия. Студентка. Область научных интересов: информационные технологии, разработка ИС. E-mail: annalob04@mail.ru.

Modular approach in information systems development: advantages and implementation

D. Rahmani, A.A. Dubrovskaya, A.A. Lobanova
Moscow Technical University of Communications and Informatics,
Moscow, Russia

Abstract: The modular architecture allows you to create scalable, flexible and easily supported information systems. The article discusses the key principles of modular design, including encapsulation, component reuse, and a microservice approach. Examples of modular systems implementation in enterprise software, web applications, and cloud platforms are analyzed. The advantages are evaluated, such as simplification of maintenance and testing, as well as possible difficulties associated with intermodular integration. The paper uses theoretical research methods, such as an analysis of existing design practices, classification of approaches according to criteria of flexibility and scalability, formalization of the principles of modular design, on the basis of which it is possible to determine how well the system architecture is designed, as well as an analogy, on the basis of which a comparison of microservice architecture and the classical modular approach is carried out. The results of the study include the systematization of module integration methods and the identification of risks of intermodular communication.

Keywords: *Modular architecture, component reuse, micro-service approach, encapsulation, scalability, module testing, inter-module integration*.
DOI: 10.14357/20790279250306 **EDN:** OTABGG

References

1. *Kotsyuba I.Y. et al.* Fundamentals of information systems design, textbook ITMO University. 2015 (In Russ).
2. *Rakhmanov V.I.* On the issue of building adaptive information systems / V. I. Rakhmanov // Innovative science. 2015;10(1):106-108 (In Russ).
3. *Golchevsky Yu.V.* The relevance of using microservices in the development of information systems / Yu.V. Golchevsky, A.V. Molenko // Bulletin of Syktyvkar University. Series 1: Mathematics. Mechanics. Computer science. 2020;2(35):25-36 (In Russ).
4. *Ignatskaya I.V.* Representation and analysis of the architecture of software systems based on the graph of interactions / I.V. Ignatskaya // Proceedings of the MAI. 2010;39:16 (In Russ).
5. *Martin R.C.* Agile Software Development: Principles, Patterns, and Practices. 2002 (In Russ).
6. Nigel Pereira finds out why big tech moved from monolithic applications to a superior form of programming in the form of microservices. Available from: https://www.sify.com/digital-transformation/why-amazon-netflix-and-uber-prefermicroservices-over-monoliths/ [Accessed April 09, 2025].
7. Introduction to Modularity and Interfaces In System Design. Available from: Introduction to Modularity and Interfaces In System Design | GeeksforGeeks [Accessed April 11, 2025].

Rahmani Jahed. Moscow Technical University of Communications and Informatics, Moscow, Russia. Senior Lecturer at the Department of Network Information Technologies and Services. Research interests: Information technology, IP development. E-mail: j.rahmani@mtuci.ru.

Alexandra A. Dubrovskaya. Student of the Moscow Technical University of Communications and Informatics, Moscow, Russia. Research interests: Information technology, IP development, big data and data analytics. E-mail: sashadubrovskay2005@gmail.com.

Anna A. Lobanova. Student of the Moscow Technical University of Communications and Informatics, Moscow, Russia. Research interests: Information technology, IP development. E-mail: annalob04@mail.ru.