

Эволюция архитектур от монолитных к модульным системам

Переход от монолитных к модульным архитектурам носит глубокие технические причины. У монолитных систем имеется **единственная точка отказа**: сбой в одном компоненте может вывести из строя всю систему ¹. Изменения и масштабирование в таком приложении требуют пересборки и перезапуска всего стека ² ³, что замедляет разработку. Крупный монолит усложняет параллельную работу больших команд и препятствует быстрому вводу новых технологий ² ⁴. В результате пострадает гибкость и скорость доставки обновлений. Чтобы избежать рисков, некоторые компании создают «ИТ-песочницы» – отдельные среды для экспериментов с новыми сервисами без влияния на боевую систему ⁴. Это иллюстрирует, насколько модулирование необходимо для безопасной и быстрой разработки.

- **Единая точка отказа и отказоустойчивость:** в монолите ошибка одного модуля останавливает всё приложение ¹ ².
- **Проблемы масштабируемости:** нельзя масштабировать только часть системы – всё приложение масштабируется целиком ².
- **Замедление разработки:** большие монолиты усложняют и замедляют выпуск новых функций ².
- **Трудности обновлений и технологий:** даже небольшое изменение требует пересборки и развёртывания всей системы ² ³, что препятствует оперативному внедрению новых технологий.

Исторические этапы перехода в ИТ

Эволюция архитектур ПО отражает эти тенденции. В эпоху **мейнфреймов (1960–1980-е)** доминировали монолитные приложения на одном большом сервере ⁵. Затем появился **клиент-сервер** (1980–90-е): вычисления разделялись между «толстым клиентом» и сервером, что стало первым шагом к распределённым системам ⁶. В 1990-е развивались **сервис-ориентированные архитектуры (SOA)** – грубые сервисы (SOAP, ESB), обеспечивавшие повторное использование, но зачастую слишком тяжеловесные ⁷.

К середине 2000-х начал набирать популярность легковесный подход: с приходом **Agile и REST** архитектуру стали дробить на мелкие сервисы. Первым известным случаем стала компания Amazon (2002): её команды начали строить независимые сервисы с собственным функционалом ⁸. В начале 2010-х Джеймс Льюис и Мартин Фаулер официально ввели термин «микросервисы» (мелкие автономные сервисы) ⁹. Одновременно рост облачных платформ (AWS, Azure) и появление Docker/Kubernetes сделали развёртывание микросервисов массовым. Классическим примером стала миграция Netflix: начиная с 2009 года компания перевела свой видеостриминг из монолитного ЦОД-а в облако и распределила более 1000 сервисов, что позволило разработчикам развёртывать код тысячи раз в день ¹⁰ ¹¹. В 2020-е возникает новое поколение паттернов – серверлесс, сервис-меш, микрофронтенды и даже возврат к «модульному монолиту» в менее критичных сценариях ¹².

Аналогичные переходы в других отраслях

Аналогичные тренды наблюдаются вне IT. В **электронике** повсеместно переходят от монолитных чипов к «чиплетам» – наборам небольших специализированных кристаллов. При изготовлении монолитного чипа брак в одном участке обнуляет весь продукт, а при использовании чиплетов дефект «отсекает» лишь один модуль, что повышает выход годных и снижает стоимость ¹³. Чиплетная архитектура обеспечивает модульность и масштабируемость: отдельные функциональные блоки (память, вычислитель, I/O) делаются отдельно и затем соединяются через стандартизованный интерфейс ¹⁴ ¹⁵. Это позволяет параллельно разрабатывать разные части и быстрее выпускать новые поколения устройств.

В **промышленных технологиях** также растёт популярность модульного подхода. Например, в автоматизации заводов широко применяются плоскостолосные шины и модульные контроллеры, что упрощает расширение систем и интеграцию новых линий. Компании создают «подключаемые» технологические блоки и даже «цифровые двойники» оборудования для гибкой сборки производств.

В **биотехнологиях** становится нормой модульный дизайн компонентов. Вместо «прямой фузии» белков учёные конструируют био-модули (например, отдельные структурные и функциональные домены). Это даёт гибкость – отдельные «блоки» можно заменять и переиспользовать ¹⁶. Как отмечено в научном блоге Ailurus, модульность в синтетических органеллах позволяет «менять компоненты без перестройки всей системы, масштабировать её, переиспользовать части» – «как строить из кубиков LEGO вместо заливки бетона» ¹⁶.

Специфика в сфере ИИ: от LLM-монолитов к мультиагентным системам

В области ИИ сохраняются сходные тенденции. Мощные LLM сами по себе представляют «монолит» – один большой невзаимодействующий компонент. Однако для сложных задач актуальны **мультиагентные системы** (MAS) на их основе. Такие системы состоят из множества автономных LLM-агентов, каждый из которых выполняет часть задачи или обладает узкими навыками ¹⁷ ¹⁸. Лингвистические модели не всемогущи – они ошибаются и не умеют параллельно решать многоэтапные задачи, требующие координации ¹⁹. В MAS агенты кооперируются, планируют, спорят или соревнуются друг с другом, приближая коллективный интеллект решениям реальных проблем ¹⁷.

Уже в 2025 году эту тенденцию называют новой индустрией: «годом AI-агентов и мультиагентности» ²⁰. Например, FinTech-ассоциация “3x10 трендов” называет AI-агентов вторым по значимости трендом после генеративного ИИ ²⁰. Стремятся внедрять готовые решения: появляются открытые продукты (LangChain, AutoGen, CamelAI и др.), которые упрощают создание мультиагентных ИИ-систем. В них каждый агент – цифровое «существо» с сенсорами, планировщиком и исполнительной частью, работающее автономно и взаимодействующее с другими агентами ²¹ ²². Такой переход от одного «большого» LLM к сети узконаправленных агентов напоминает общий тренд декомпозиции сложных систем на независимые блоки.

Архитектурные преимущества модульного подхода

Модульность даёт ряд технических преимуществ. Во-первых, **отказоустойчивость**: сбой одного модуля не парализует всю систему – остальные сервисы продолжают работать ²³. Это критично

для приложений с высокими требованиями к надёжности. Во-вторых, **масштабируемость**: каждый модуль можно масштабировать по отдельности. Нагруженный сервис разворачивают на дополнительных узлах, не затрагивая другие ²³ ²⁴. Третье – **гибкость и быстрота обновления**. Компоненты с чётко выделенными границами позволяют вносить локальные изменения без остановки всего приложения ²⁴ ²⁵. Например, Atlassian отмечает, что переход на микросервисы позволил им увеличить частоту релизов с одного раза в неделю до нескольких раз в день ²⁴. В-четвёртых, **повторное использование**: стандартизированные модули легко адаптируются к новым задачам ¹⁶. Как сказано о модульном дизайне в биотехнологии, это «гибко, многократно и надёжно» – идеальная основа для инноваций ¹⁶. Наконец, **автономия команд**: небольшие разработческие группы могут отвечать за свой модуль и развивать его независимо, что улучшает управляемость разработки и ускоряет выпуск новых функций ²⁴ ¹⁶.

Схематически разница видно на примере ПО: монолит – единое целое с одной точкой входа, тогда как микросервисы – набор отдельных компонентов. Ниже изображён пример монолитной архитектуры как единый процесс и единую базу данных ²⁶.

²⁶ Монолитная архитектура представляет собой единую программу с одной базой кода. Такой подход упрощает начальную разработку ²⁶, но усложняет масштабирование, развертывание и обновление, поскольку даже небольшое изменение требует пересборки всего приложения ² ³. Модульная архитектура решает эти проблемы разделением системы на независимые блоки.

Открытые модели и мультиагентные фреймворки – новая волна AI-платформ

Современный «стек ИИ» всё больше основан на открытых решениях. Платформы вроде Hugging Face выпускают **открытые предобученные модели** и инструменты для их дообучения (fine-tuning) под конкретные задачи ²⁷. Это снижает барьеры входа: доступ к передовым LLM уже не требует ресурсов Big Tech – невелико количество «топ-моделей» и у небольших компаний появились свои специализированные (SLM) и даже кооперативные AI-агенты ²⁷. Открытые модели позволяют сменить ядро ИИ-системы без перестройки всей инфраструктуры ²⁸, что ускоряет инновации и демократизирует технологию.

Появление **фреймворков для мультиагентных систем** ускоряет создание сложных AI-приложений. Так, LangChain обеспечивает создание цепочек вызовов LLM, интеграцию с данными и разработку диалоговых агентов ²¹. Microsoft AutoGen – открытая платформа для мультиагентных систем, где агенты на базе LLM взаимодействуют друг с другом для совместного решения задач ²². Есть и другие проекты (CamelAI, Haystack, DeepPavlov и др.), упрощающие оргпроцессы в разработке ИИ-агентов. Наличие этих открытых решений стимулирует формирование экосистем и ускоряет появление новых продуктов на базе ИИ ²⁸ ²¹.

Влияние на разработку, затраты, конкуренцию и инновации

Архитектурная трансформация сильно меняет рынок ИТ. Для разработки модульность означает повышение скорости выпуска и большую гибкость. Известно, что микросервисная архитектура Netflix позволила его команде кодировать и деплоить изменения **тысячи раз в день** ¹¹ ¹⁰. Благодаря этому сокращается время выхода фичи на рынок. Одновременно **снижается риск и стоимость простоев**: отказоустойчивые системы требуют меньших вложений в аварийное восстановление и уменьшают убытки от простоев (у крупных игроков каждая минута простоя стоит тысячи долларов ²⁹).

С точки зрения экономики, переход к модулям часто требует изначальных инвестиций (рефакторинг, настройка CI/CD, оркестрация), но затем даёт экономию: повторно используемые блоки и малые независимые сервисы легче сопровождать и развивать. Открытые решения дополнительно снижают затраты – небольшие компании теперь могут внедрять передовые ИИ без создания моделей «с нуля» ²⁷. Это размывает барьеры входа и усиливает конкуренцию: игры крупного и малого бизнеса становятся более равноправными. Многие видят, что доступ к базовым моделям, а затем настройка своих агентов, превращаются в массовую практику.

В итоге модульная трансформация ускоряет инновации. Команды экспериментируют локально и быстро интегрируют новые разработки без риска обрушить систему в целом ²⁴ ²⁸. Гибкость архитектуры позволяет быстро адаптироваться к новым требованиям рынка и корректировать направление развития продукта. Ускоренная разработка и открытые сообщества вместе создают эффект «экосистемы»: новые идеи быстрее проверяются и масштабируются.

Заключение

Переход от монолитных к модульным архитектурам – устойчивый тренд в IT и других отраслях. Технические преимущества модульности (гибкость, масштабируемость, отказоустойчивость, повторное использование) и доступность открытых платформ делают этот подход доминирующим при создании сложных систем. В сфере ИИ это проявляется в переходе от единой LLM-модели к системам из множества специализированных агентов ¹⁷ ²⁷. Ожидается, что такая эволюция ускорит темп инноваций, приведёт к снижению издержек и расширению конкуренции, а также позволит более эффективно внедрять передовые технологии в разных областях ²⁷ ²⁹. По мере дальнейшего развития ИИ и цифровой экономики модульные архитектуры будут сохранять ключевую роль, обеспечивая компаниям необходимую скорость и гибкость.

Источники: Анализ современных исследований и публикаций ¹ ⁵ ³⁰ ¹⁷ ¹¹ ²³ ¹⁴ ¹⁶ ²¹ ² и др.

¹ Переход от монолита к микросервисам: история и практика / Хабр

<https://habr.com/ru/companies/raiffeisenbank/articles/458404/>

² ³ ¹¹ ²⁶ Сравнение микросервисной и монолитной архитектур | Atlassian

<https://www.atlassian.com/ru/microservices/microservices-architecture/microservices-vs-monolith>

⁴ Здоровая цифровизация

<https://www.skolkovo.ru/expert-opinions/zdorovaya-cifrovizaciya/>

⁵ ⁶ ⁷ ⁸ ⁹ ¹² Microservices - history - follow the idea - Obsidian Publish

<https://publish.obsidian.md/followtheidea/Content/AI/Microservices+-+history>

¹⁰ Microservices vs. monolithic architecture | Atlassian

<https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith>

¹³ ¹⁴ ¹⁵ Why chiplets will transform electronic system design - DENA

<https://www.designing-electronics.com/why-chiplets-will-transform-electronic-system-design/>

¹⁶ Modularity Is All You Need

<https://www.ailurus.bio/post/fantastic-organelles-and-how-to-build-them-chapter-4--modularity>

¹⁷ Мультиагенты, основанные на больших языковых моделях(LLM) / Хабр

<https://habr.com/ru/articles/796555/>

18 19 20 Мультиагентные системы на основе LLM: Как это работает и зачем нужно

<https://cloud.ru/blog/multiagentnyye-sistemy-na-osnove-llm>

21 22 Лучшие AI фреймворки для агентов с открытым исходным кодом: какой выбрать?

<https://external.software/archives/19262>

23 24 25 29 5 Advantages of Microservices [+ Disadvantages] | Atlassian

<https://www.atlassian.com/ru/microservices/cloud-computing/advantages-of-microservices>

27 28 30 Новый стек ИИ: интеграция и масштабирование ИИ-решений с помощью модульной архитектуры

<https://www.itweek.ru/ai/article/detail.php?ID=230731>