ГЛАВА 1

Введение

Итак, вы интересуетесь архитектурой программного обеспечения. Возможно, вы — разработчик, желающий подняться на следующую ступень своей карьеры. Или менеджер проекта, который хочет понять, как работает архитектура. А может, вы — «архитектор по воле случая»: специалист, которому приходится принимать архитектурные решения (см. ниже), но который не занимает должность архитектора... пока.

Для чего погружаться в мир архитектуры ПО? Например, если у вас накопился опыт работы над множеством проектов и вы хотите глубже понять, как взаимодействуют более крупные компоненты системы и на какие компромиссы пришлось идти при проектировании. В таком случае архитектура становится очевидным следующим шагом карьерного роста.

Эта книга написана для всех вас. Она описывает чрезвычайно многогранную роль архитектора программного обеспечения.

Архитектор ПО должен хорошо понимать и тщательно анализировать программные системы во всей их сложности, а также принимать важные компромиссные решения, иногда не владея полной информацией. Многие разработчики, обеспокоенные тем, что генеративный ИИ сможет постепенно лишить их работы, подумывают переключиться на архитектуру — заменить эту роль будет намного сложнее. Архитекторы принимают именно такие решения, которые не может принять ИИ, оценивая различные компромиссы в сложных, изменяющихся контекстах.

Архитектуру, как и многие произведения искусства, можно понять только в контексте. Архитектор принимает решения с учетом реалий среды. Например, основной целью архитектуры ПО конца XX века было более эффективное использование общих ресурсов, поскольку вся инфраструктура того времени состояла из дорогих коммерческих продуктов: операционных систем, серверов приложений, серверов баз данных и т. д.

Представьте, что вы приходите в дата-центр образца 2002 года и говорите руководителю команды эксплуатации: «У меня отличная идея — принципиально новая архитектура, в которой каждый сервис работает на отдельном изолирован-

ном оборудовании с собственной базой данных. Мне понадобится 50 лицензий Windows, еще 30 лицензий сервера приложений и как минимум 50 лицензий сервера базы данных». Сегодня мы можем строить такие архитектуры только благодаря развитию продуктов с открытым исходным кодом и усовершенствованным инженерным практикам DevOps. Все архитектуры являются продуктом своего контекста — помните об этом, читая книгу.

Определение программной архитектуры

Что же считать программной архитектурой? На рис. 1.1 показано, с каких позиций необходимо ее рассматривать. Определение включает четыре измерения. Архитектура программной системы состоит из *архитектурного стиля* (отправная точка) в совокупности с необходимыми *свойствами*; *логическими компонентами*, реализующими ее поведение; и наконец, *архитектурными решениями*, которые все это подкрепляют. Структура системы обозначается жирными черными линиями. Сейчас мы кратко рассмотрим эти измерения в том порядке, в котором их анализируют архитекторы, а подробности раскроем в следующих главах.

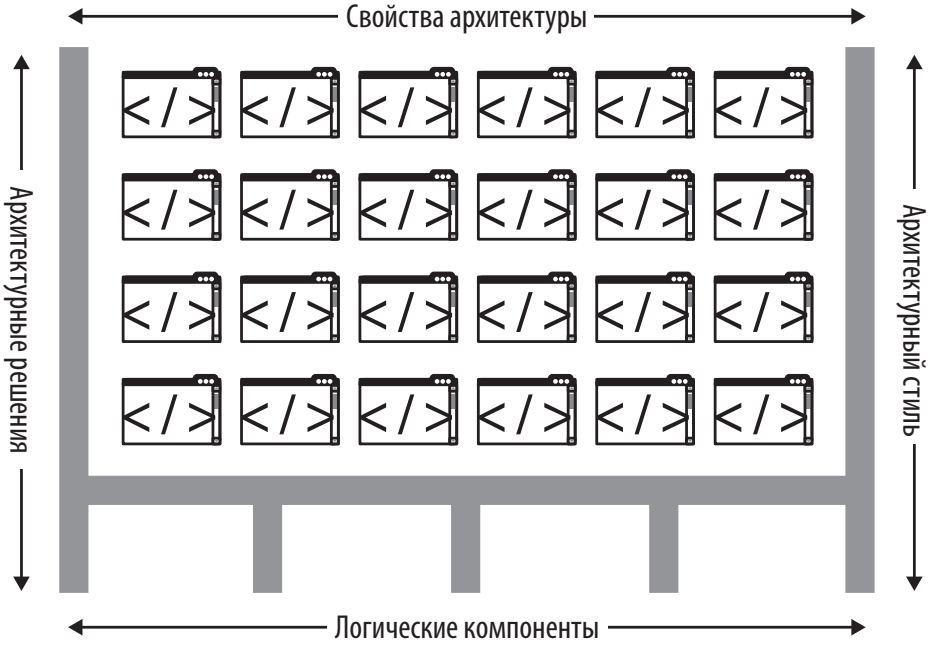

Рис. 1.1. Архитектура состоит из структуры системы, объединенной со свойствами архитектуры (которые обозначаются словами с окончанием «-ость»), логическими компонентами, архитектурными стилями и решениями

Свойства архитектуры (рис. 1.2) определяют ее *возможности* и служат критериями ее успеха — короче, это все, что система должна *делать*. Свойства архитектуры играют настолько важную роль, что их определению и пониманию посвящено несколько глав этой книги.

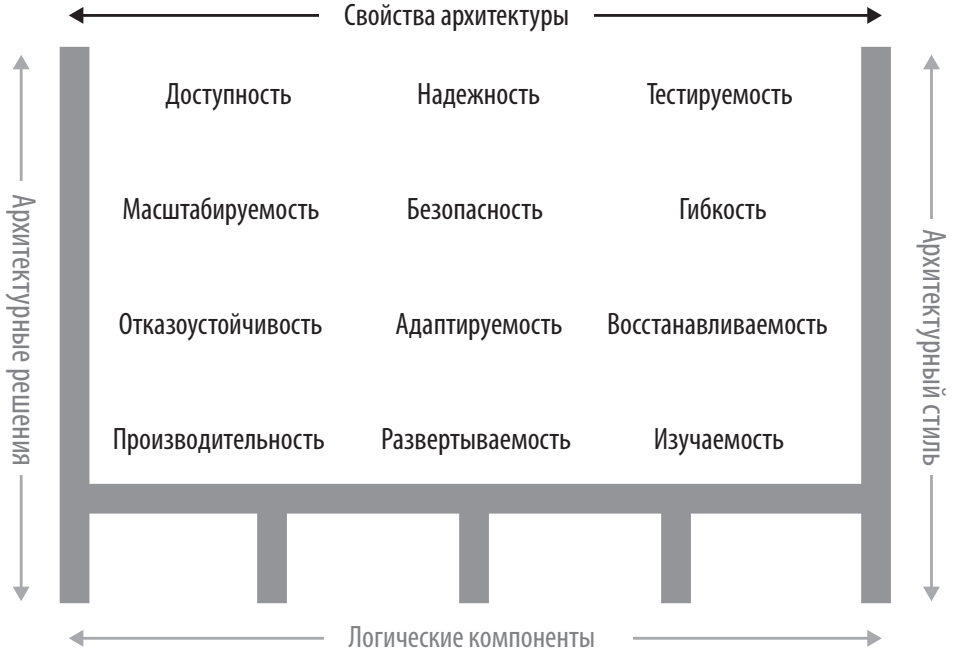

Рис. 1.2. Свойства архитектуры относятся к тому, что должно поддерживаться системой и описывается словами с окончанием «-ость»

Если свойства архитектуры определяют возможности системы, то *логические компоненты* определяют ее *поведение*. Проектирование логических компонентов — одна из ключевых структурных операций, выполняемых архитекторами. На рис. 1.3 логические компоненты образуют домены, сущности и рабочие процессы приложения.

После того как архитектор проанализирует свойства архитектуры и логические компоненты, необходимые системе (подробнее об этом позже), он будет знать достаточно для выбора подходящего архитектурного стиля как отправной точки на пути реализации решения.

Четвертым измерением, определяющим архитектуру ПО, являются *архитектурные решения*. Ими определяются правила, по которым должна выстраиваться система. Например, архитектор может решить, что доступ к базе данных (БД) в многоуровневой архитектуре должен происходить только с бизнес-уровня

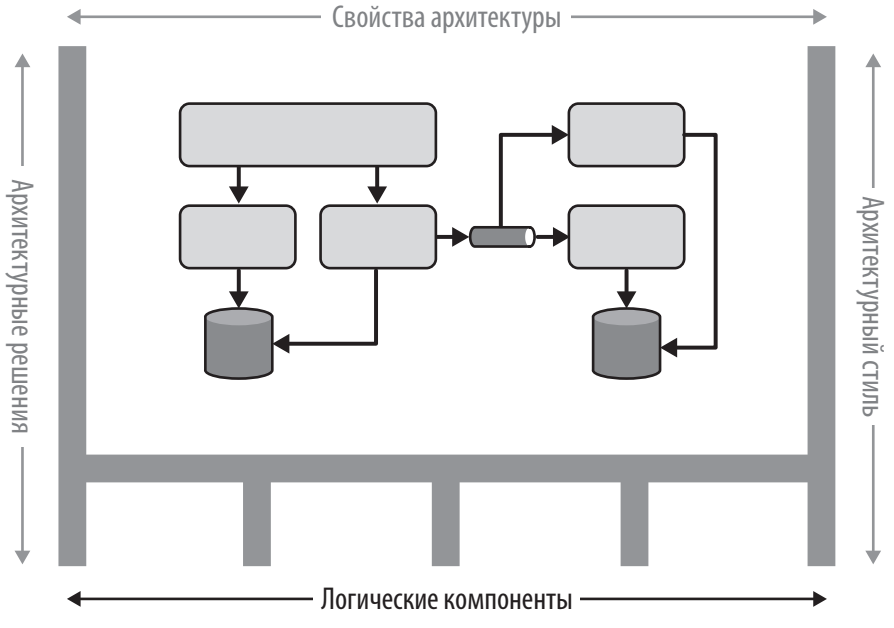

Рис. 1.3. Логические компоненты структурируют поведение системы

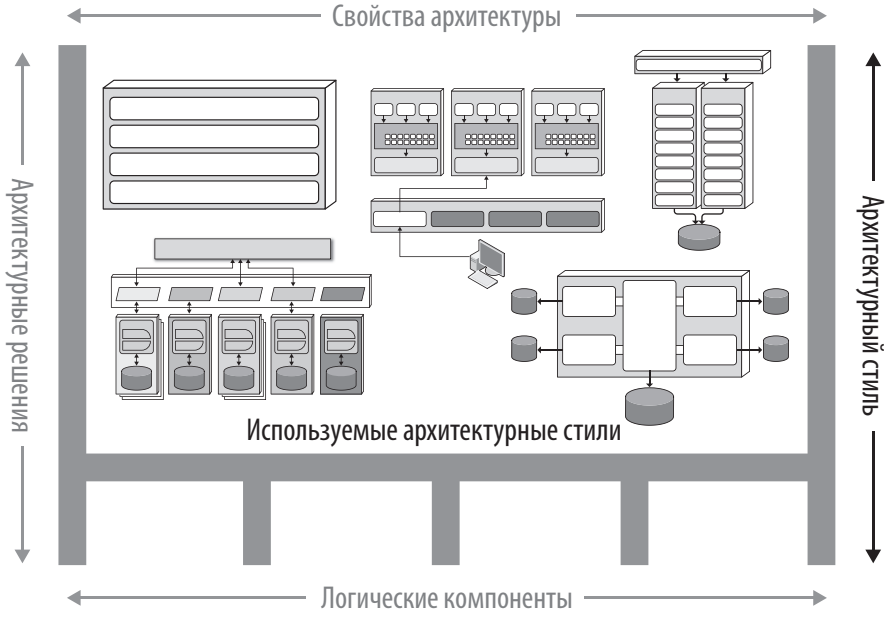

Рис. 1.4. Выбор архитектурного стиля подразумевает поиск простейшего пути реализации для заданного набора требований

и уровня сервисов (рис. 1.5), а прямые обращения к ней с уровня представления должны быть запрещены[1]. Архитектурные решения задают ограничения системы и формируют у команд разработчиков понимание, что разрешено, а что нет.

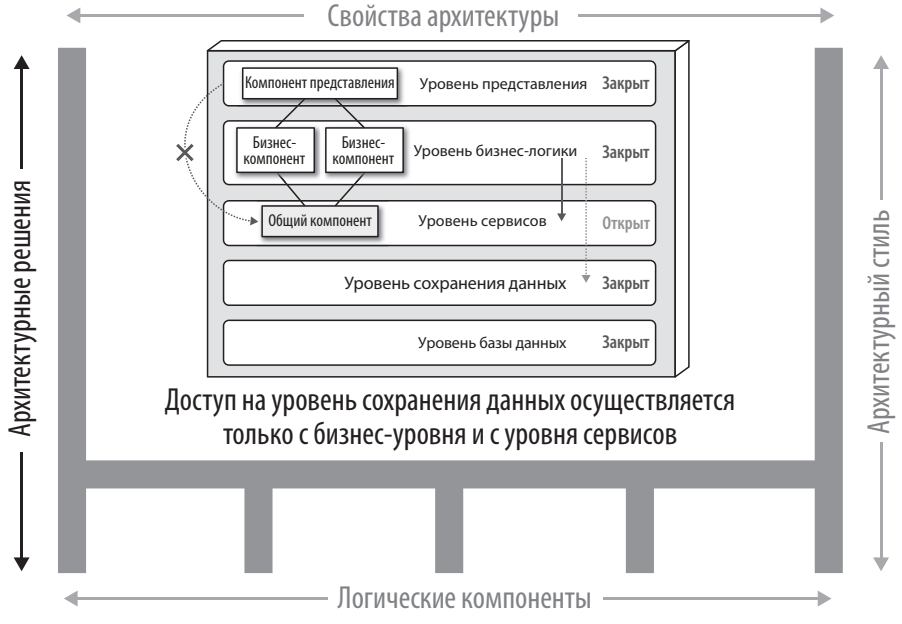

Рис. 1.5. Архитектурные решения являются правилами построения систем

Архитектурные решения и способы их документирования подробно рассматриваются в главе 21.

Законы архитектуры ПО

Когда авторы начали работать над первым изданием этой книги, они руководствовались высокой целью: найти принципы, всегда справедливые для программных архитектур, и оформить их в виде «законов» программной архитектуры. Во время всей работы над книгой мы постоянно старались найти эти непреложные истины. Мы надеялись найти 10, возможно, 15 таких законов. К нашему удивлению, мы выявили всего два закона в первом издании и обнаружили еще один во время работы над вторым. В полном соответствии

[1] В технической документации и профессиональной литературе по архитектуре ПО термин *layer* переводится и как «слой», и как «уровень», соответственно, «многослойная» или «многоуровневая» архитектура. Более часто встречается термин «слой», но для преемственности с первым изданием в книге оставлен «уровень». — *Примеч. науч. ред.*

с исходными намерениями эти три закона можно назвать универсальными и открывающими множество важных перспектив для практикующих архитекторов.

С Первым законом программной архитектуры мы постоянно сталкивались в своей работе. Пожалуй, он добирается до сути того, почему эти универсальные истины кажутся нам такими неуловимыми:

> Все в архитектуре ПО — это компромиссы.
>
> — *Первый закон программной архитектуры*

Ничто не существует в четких, удобных рамках. Каждое решение архитектора ПО должно учитывать множество переменных, которые принимают разные значения в зависимости от обстоятельств. Компромиссы — сущность архитектурных решений.

> Если вы думаете, что обнаружили что-то, не предполагающее компромисса, то скорее всего, вы просто *не обнаружили* этот компромисс... пока.
>
> — *Следствие 1*

> Невозможно провести анализ компромиссов один раз и забыть о нем навсегда.
>
> — *Следствие 2*

Команды любят стандарты. Было бы удобно, если бы архитекторы могли провести Один Большой Совет по Компромиссам и раз и навсегда определить параметры используемого стиля, способы взаимодействия разных частей архитектуры, управления общей функциональностью, а также принять множество других решений на будущее. Но это невозможно, потому что каждая ситуация заставляет нас заново переоценивать все эти компромиссы. (Мы видели команды, которые пытались упростить все, например, решили по умолчанию использовать только хореографию в распределенных процессах — и выяснили, что иногда это срабатывает, а иногда оборачивается настоящим эпичным провалом. См. раздел «Хореография и оркестрация» на с. 373.)

Архитектура не сводится к комбинации структурных элементов, поэтому наше многомерное решение включает принципы, свойства и т. д. Этот факт отражен во Втором законе программной архитектуры:

> *«Почему»* важнее, чем *«как»*.
>
> — *Второй закон программной архитектуры*

Если мне как опытному архитектору кто-то покажет архитектуру, которая еще никогда мне не встречалась, я смогу разобраться в том, *как* она работает, но возможно, я не сразу пойму, *почему* предыдущий архитектор или команда приняли те или иные решения. Архитекторы принимают решения в высшей степени конкретных контекстах, что усложняет их обобщение и масштабирование. Чтобы понять, почему архитектор принял то или иное решение, необходимо понимать,

какие компромиссы он рассматривал; эта информация дополняет контекст выбора одного решения вместо другого. Одна из определяющих особенностей решений архитектур ПО заключается в том, что они редко бывают бинарными. Мы приходим к Третьему закону программной архитектуры:

> Многие архитектурные решения не бинарны. Они лежат между крайними точками.
>
> — *Третий закон программной архитектуры*

В этой книге мы подчеркиваем, *почему* архитекторы принимают те или иные решения, и описываем присущие им компромиссы. Также в главе 21 будут описаны полезные приемы для фиксации важных решений.

В книге читатели увидят, как работают эти законы, и мы рекомендуем помнить о них при оценке архитектурных решений. Мы еще вернемся к ним в главе 27, где будут приведены дополнительные примеры.

Итак, у нас есть рабочее определение программной архитектуры, и мы можем перейти к самой роли.

Ожидания от работы архитектора

Определить роль архитектора ПО ничуть не легче, чем дать определение самой архитектуре. Его обязанности могут варьироваться от задач опытного разработчика до специалиста, формирующего технологическую стратегию компании. Так что вместо того, чтобы тратить время на попытку определения этой роли сконцентрируемся на *ожиданиях* от работы архитектора. Независимо от конкретных полномочий, титулов или должностных обязанностей, от архитектора ПО ожидают:

- принятия архитектурных решений;
- постоянного анализа архитектуры;
- следования современным тенденциям;
- контроля за выполнением принятых решений;
- обладания обширными знаниями и опытом;
- компетентности в конкретной области бизнеса;
- владения навыками межличностного общения;
- четкого понимания политики компании.

Эффективная и плодотворная работа в должности архитектора программного обеспечения невозможна без понимания каждого из этих ожиданий и соответствия им. В этом разделе мы рассмотрим каждое из них более подробно.

Принятие архитектурных решений

От архитектора ожидают разработки архитектурных решений и определения принципов проектирования, которые будут служить руководством для принятия технологических решений в рамках команды, отдела или всей организации.

Ключевым здесь является слово *руководство*. Архитектор должен *направлять*, а не определять выбор технологических решений. Например, архитектор может решить, что для разработки внешнего интерфейса нужно использовать библиотеку React.js. В этом случае он принимает технологическое решение, а надо бы — архитектурное: определить принцип проектирования, который поможет команде разработчиков сделать свой выбор. Архитектор должен рекомендовать команде использовать для веб-разработки внешнего интерфейса реактивно-ориентированную среду и тем самым подвести к выбору между Angular, Elm, React.js, Vue или любым другим фреймворком на реактивной основе. Ключом к принятию эффективного архитектурного решения является вопрос о том, помогает ли оно направлять команды к выбору правильных технологий или же такой выбор делается за них. И тем не менее бывает так, что для сохранения конкретных свойств, например масштабируемости, производительности или доступности, архитектору приходится принимать те или иные технологические решения. В таком случае оно все равно будет считаться архитектурным, даже если определяет конкретную технологию. Архитекторы часто испытывают трудности с поиском правильного курса, поэтому архитектурным решениям целиком посвящена глава 21.

Постоянный анализ архитектуры

Архитектор должен постоянно анализировать архитектуру и текущую технологическую среду и рекомендовать решения по их совершенствованию.

Это позволяет поддерживать *жизнеспособность архитектуры*, то есть оценивать, насколько архитектура, определенная года три (или более) назад, актуальна на *сегодняшний день* с учетом изменений как в бизнесе, так и в технологиях. Исходя из нашего опыта, далеко не все архитекторы уделяют достаточно внимания постоянному анализу существующих архитектур. В результате большинство архитектур подвергается структурному распаду, происходящему из-за изменений, вносимых разработчиками в код или проект, что влияет на требуемые свойства архитектуры, например на производительность, доступность или масштабируемость.

Другие особенности, о которых часто забывают архитекторы, — среды тестирования и выпуска программного продукта. Возможность гибкой модификации кода дает очевидные преимущества, но если командам разработчиков требуются недели на тестирование продукта и месяцы на его релиз, значит, архитекторам не удалось достичь гибкости архитектуры в целом.

Чтобы убедиться в состоятельности архитектуры, архитектор должен проводить комплексный анализ изменений заданной технологии и предметной области. Хотя в объявлениях о вакансии подобное требование к уровню квалификации фигурирует довольно редко, архитектор все же должен соответствовать и этому ожиданию.

Следование современным тенденциям

Архитектор должен быть в курсе последних тенденций в технологии и отрасли.

Чтобы сохранять свою востребованность (и не потерять работу!), разработчики должны быть в курсе технологических новинок, относящихся к их повседневной деятельности. К архитектору предъявляются еще более жесткие требования: он должен следить за текущими изменениями не только в технологии, но и в своей отрасли. Принимаемые архитектором решения носят долговременный характер и трудно поддаются изменениям. Четкое понимание основных тенденций помогает архитектору подготовиться к будущим вызовам и принимать правильные решения. Например, несколько лет назад квалифицированный архитектор должен был хорошо разбираться в теме облачных хранилищ и развертывания, а пока мы готовили второе издание книги, генеративный ИИ стал оказывать колоссальное влияние на многие вопросы разработки.

Отслеживание тенденций и следование им даются нелегко, особенно архитектору ПО. Различные технологии и ресурсы, позволяющие успешно решать эту задачу, будут рассмотрены в главе 2.

Контроль за выполнением принятых решений

Архитектор должен обеспечивать соответствие архитектурным решениям и принципам проектирования.

Контроль за выполнением принятых решений означает, что архитектор обязан постоянно проверять, что разработчики следуют принятым, задокументированным и доведенным до них архитектурным решениям и принципам проектирования.

Представьте, что вы как архитектор принимаете решение, что в многоуровневой архитектуре доступ к базе данных возможен только с уровней бизнеса и сервисов (и невозможен с уровня представления). Следовательно (как будет показано в главе 10), даже для самых простых обращений к базе данных уровень представления должен пройти через все уровни архитектуры. Разработчик пользовательского интерфейса может не согласиться с этим и с целью повышения производительности обращаться к базе данных (или к уровню постоянного хранения данных) напрямую. Но архитектор принял данное решение по

конкретной причине: чтобы изменения в базе данных не затрагивали уровень представления. Если не обеспечить соблюдение архитектурных решений, могут быть допущены такие нарушения, при которых архитектура перестанет отвечать установленным требованиям, а приложение или система не будут работать, как ожидалось. В главе 6 мы подробнее поговорим об оценке соблюдения решений и принципов с помощью автоматизированных функций пригодности и соответствующих автоматизированных инструментов.

Обширные знания и опыт

Ожидается, что архитектор обладает опытом работы с многочисленными и разнообразными технологиями, средами разработки, платформами и средами окружения.

Это не означает, что архитектор должен быть специалистом по каждому фреймворку, платформе и языку, скорее он должен быть знаком с разнообразием технологий. Большинство современных сред имеет неоднородный характер, и архитектор должен по крайней мере знать возможности взаимодействия нескольких систем и сервисов независимо от используемых в них языков, платформ и технологий.

Лучший способ соответствовать этому требованию — постоянно выходить из зоны комфорта, активно накапливать опыт использования различных языков программирования, платформ и технологий. Архитектор должен *расширять*, а не углублять свои технические знания. Технический кругозор включает то, что вам знакомо в общих чертах, в сочетании с тем, в чем вы хорошо разбираетесь. Например, для архитектора намного более ценно знать 10 разных инструментов кэширования со всеми их достоинствами и недостатками, чем быть узким специалистом по одному из них.

Компетентность в конкретной области бизнеса

Ожидается, что архитектор достаточно хорошо ориентируется в конкретной области бизнеса.

Успешные архитекторы программного обеспечения разбираются не только в технологии, но и в области бизнеса, относящейся к решаемым задачам. Без знаний в *бизнес-сфере* трудно определить круг решаемых задач, цели и требования, а это затрудняет разработку оптимальной архитектуры, отвечающей потребностям бизнеса. Предположим, что архитектор в крупном финансовом учреждении не разбирается в общих финансовых понятиях: например, не знает, что такое *средний индекс направленности*, *алеаторные сделки*, *стабильный рост цен* или *неприоритетный долг*. Без этих знаний архитектор не сможет общаться со стейкхолдерами и бизнес-пользователями и быстро утратит доверие.