ГЛАВА 1

Введение

Специальность «архитектор программного обеспечения» находится в верхних строчках многочисленных рейтингов вакансий по всему миру. Но когда читатели смотрят на *другие* специальности в этих списках (например, фельдшер или экономист), то видят вполне определенный путь профессионального роста. Так почему же развитие карьеры архитектора программного обеспечения мало кому понятно?

Во-первых, в самой профессиональной среде отсутствует четкое определение архитектуры программного обеспечения (ПО). При чтении установочных лекций студенты часто просят дать краткое описание того, чем занимается архитектор ПО, но мы отвечаем уклончиво. И не мы одни. В своей знаменитой статье «Who Needs an Architect?» («Кому нужен архитектор?»)[1] Мартин Фаулер (Martin Fowler), как известно, отказался от попытки дать четкое определение соответствующему понятию, вернувшись вместо этого к известной цитате:

> Архитектура — это о важном... что бы это ни было.
>
> *Ральф Джонсон (Ralph Johnson)*

Однажды нам все же пришлось создать ментальную карту (майндмэп), показанную на рис. 1.1, которая, к сожалению, не является исчерпывающей, но при этом показывает масштаб архитектуры ПО. Через некоторое время мы предложим свое определение этому понятию.

Во-вторых, как показано на схеме, зона ответственности архитектора ПО весьма велика и постоянно расширяется. Лет десять назад архитекторы программного обеспечения занимались чисто техническими аспектами архитектуры: модульностью, компонентами и паттернами. Но благодаря появлению новых архитектурных стилей с более широким спектром возможностей (например,

[1] https://martinfowler.com/ieeeSoftware/whoNeedsArchitect.pdf

микросервисов, обязанности архитектора значительно увеличились. Множество пересечений архитектуры с остальными организационными задачами будет рассмотрено в следующем далее разделе «Пересечение архитектуры и...» (с. 37).

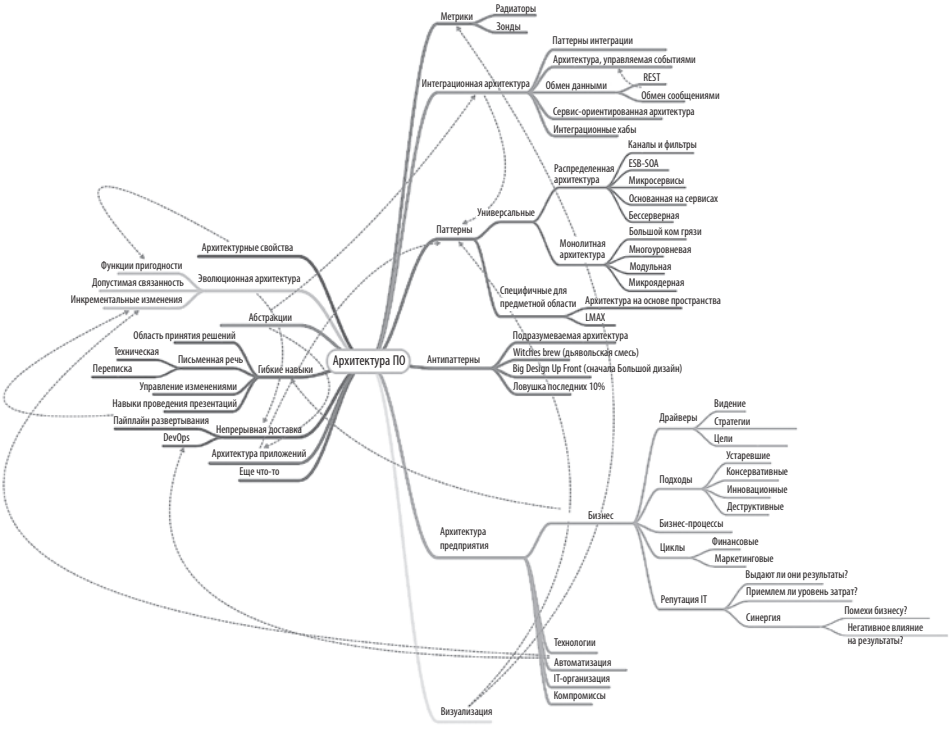

Рис. 1.1. Архитектор ПО должен обладать техническими знаниями, гибкими навыками (soft skills), а также понимать бизнес-процессы и множество других вещей

В-третьих, архитектура ПО — это постоянно меняющаяся концепция, потому что экосистема разработки программного обеспечения слишком быстро развивается. Любое принятое сегодня определение через несколько лет безнадежно устареет. Определение архитектуры ПО в Википедии дает вполне приемлемое общее представление, но многие положения уже неактуальны, например: «Архитектура программного обеспечения относится к выбору фундаментальных структурных решений, внесение изменений в которые после их реализации обходится слишком дорого». Но с тех пор специалисты разработали современные архитектурные стили (микросервисы) с идеей поэтапного встраивания, при котором внесение структурных изменений уже не обходится «слишком дорого». Конечно, эта возможность означает компромисс с другими вопросами,

например связанностью. Во многих книгах по программной архитектуре она представляется чисто статической, и считается, что после однократного внедрения ее можно спокойно игнорировать. Однако неизбежная динамическая природа архитектуры ПО, да и самого определения этого понятия прослеживается во всей книге.

В-четвертых, основной объем сведений об архитектуре ПО несет чисто историческую нагрузку. Читателям страницы Википедии, несомненно, бросится в глаза множество сокращений и перекрестных ссылок на целую вселенную знаний. Но многие из этих сокращений, по сути, устарели или не прижились. Даже те методы, которые несколько лет назад представлялись правильными, сейчас не работают, потому что изменился контекст. История развития архитектуры ПО насыщена принятыми решениями, которые в дальнейшем привели к разрушительным последствиям. Многие из этих уроков мы рассмотрим здесь. Так почему же эта книга по основам архитектуры программного обеспечения появилась именно сейчас? В нашем мире все постоянно меняется, и область архитектуры ПО не исключение. Новые технологии, методы, возможности... по сути, в последнем десятилетии проще найти что-то, что не изменилось, чем перечислить все изменения. И архитекторы программного обеспечения вынуждены принимать решения в этой постоянно меняющейся экосистеме. Поскольку изменения касаются практически всего, включая и сами основы принятия наших решений, архитекторы должны каждый раз пересматривать ряд аксиом, на которых базировались предыдущие работы. Например, в ранее вышедших книгах об архитектуре ПО не рассматривалось влияние методологии DevOps, поскольку на момент написания этих книг ее просто не было.

Изучая архитектуру, нужно осознавать, что, как и во многих других искусствах, ее премудрости можно усвоить только в контексте. Многие решения принимаются архитекторами на основе особенностей той среды, в которой они оказались. Например, основной целью архитектуры ПО конца XX века было достижение более эффективного использования 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 начальнику отдела эксплуатации: «У меня есть отличная идея по внесению кардинальных изменений в сам стиль архитектуры: запуск каждого сервиса будет на отдельной, предназначенной только для него машине, с его собственной выделенной базой данных». (Это суть того, что нам сейчас известно как микросервисы.) «И для этого мне понадобятся 50 лицензий на Windows, 30 лицензий на сервер приложения и как минимум 50 лицензий на сервер базы данных». В 2002 году попытка создания архитектуры, похожей на микросервисы, обошлась бы слишком дорого. А вот

с появлением в последние годы открытого исходного кода в сочетании с новой революционной практикой разработки и сопровождения программных продуктов в рамках методологии DevOps построение именно такой архитектуры вполне оправданно. Читатели должны понимать, что все архитектуры являются продуктом конкретных обстоятельств.

Определение архитектуры программного обеспечения

Многие специалисты пытались дать точное определение понятию «архитектура программного обеспечения». Одни считали, что архитектура ПО является по своей сути неким *планом системы*, а другие определяли ее как *дорожную карту* разработки системы. Но эти общие определения не отвечали на вопрос о конкретном содержимом плана или карты. Например, что именно архитектор должен исследовать при анализе архитектуры?

На рис. 1.2 показан способ осмысления архитектуры ПО. Согласно этому определению, она состоит из *структуры* системы (показанной широкими серыми линиями, поддерживающими архитектуру) в сочетании со *свойствами архитектуры* (выражаемыми словами с окончанием *-ость*), которые должны поддерживаться системой, *архитектурными решениями* и, наконец, *принципами проектирования*.

Под *структурой* системы (рис. 1.3) понимается тип архитектурного стиля (или стилей), в котором реализована система (например, микросервисы, многоуровневая система или же микроядро). Составить полное представление об архитектуре исключительно по структуре невозможно. Предположим, к примеру, что архитектора попросили дать описание архитектуры, а он ответил: «Это архитектура микросервисов». В этом случае он рассказал только лишь о *структуре* системы, но не об ее *архитектуре*. Для полного представления об архитектуре системы нужны также ее свойства, архитектурные решения и принципы проектирования.

Еще одним элементом определения архитектуры ПО являются ее свойства (рис. 1.4). Свойства архитектуры определяют критерии ее успеха, не имеющие ничего общего с функциональными возможностями системы. Заметьте, что все перечисленные свойства не требуют никаких сведений о функциональности системы и в то же время нужны для ее корректной работы. Свойства архитектуры играют настолько важную роль, что их определению и пониманию посвящено несколько глав этой книги.

Рис. 1.2. Архитектура состоит из структуры в сочетании со свойствами (выражаемыми словами с окончанием *-ость*), архитектурными решениями и принципами проектирования

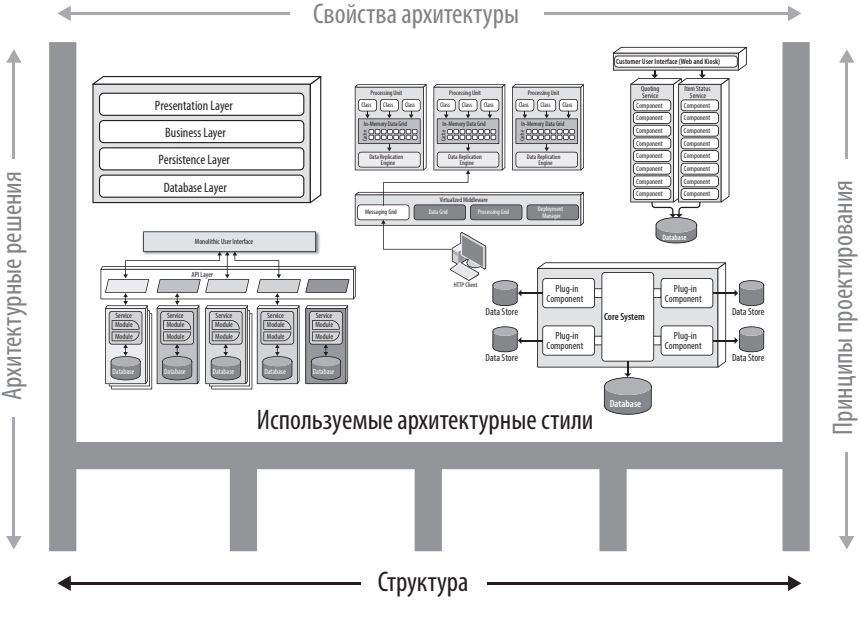

Рис. 1.3. Под структурой подразумевается тип архитектурных стилей, используемых в системе

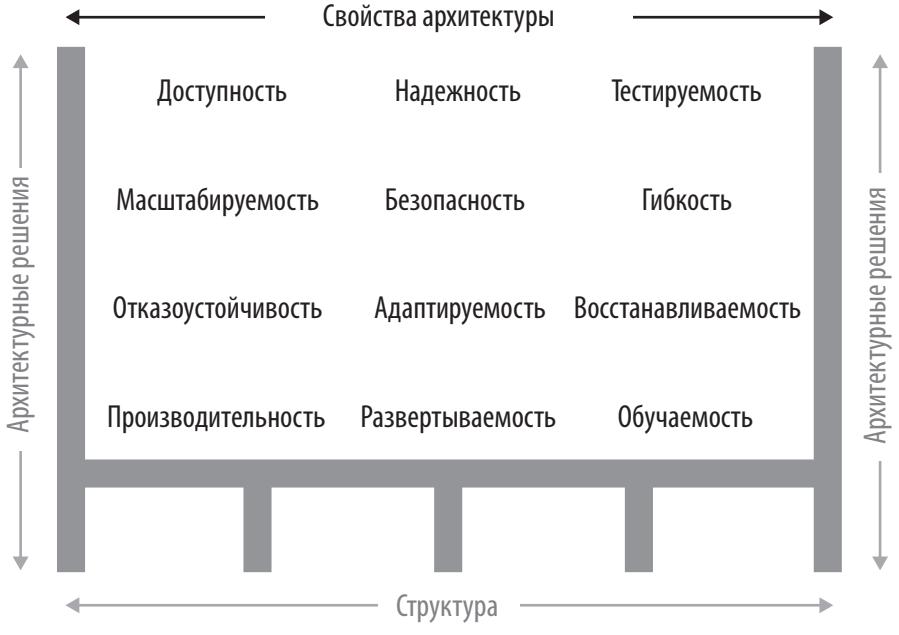

Рис. 1.4. Свойства архитектуры относятся к тому, что должно поддерживаться системой и описывается словами с окончанием *-ость*

Следующим фактором, определяющим архитектуру ПО, являются *архитектурные решения*. Ими определяются правила, по которым должна выстраиваться система. Например, архитектор должен решить, что доступ к базе данных (БД) в многоуровневой архитектуре должен быть только с бизнес-уровня и с уровня сервисов (рис. 1.5), а прямые обращения к ней с уровня представления должны быть запрещены. Архитектурные решения формируют ограничения системы и дают командам разработчиков понять, что разрешено, а что нет.

Если из-за какого-то условия или другого ограничения конкретное архитектурное решение не может быть реализовано в одной из частей системы, это решение (или правило) может быть нарушено посредством так называемого *отступления (variance)*. Модели отступлений, используемые наблюдательным советом за архитектурой (architecture review board, ARB) или главным архитектором, имеются в большинстве организаций. Эти модели определяют процесс поиска отступления от конкретного стандарта или архитектурного решения. Исключение из конкретного архитектурного решения анализируется советом ARB (или главным архитектором, если такого совета нет) и либо утверждается, либо отклоняется на основе обоснований и компромиссов.

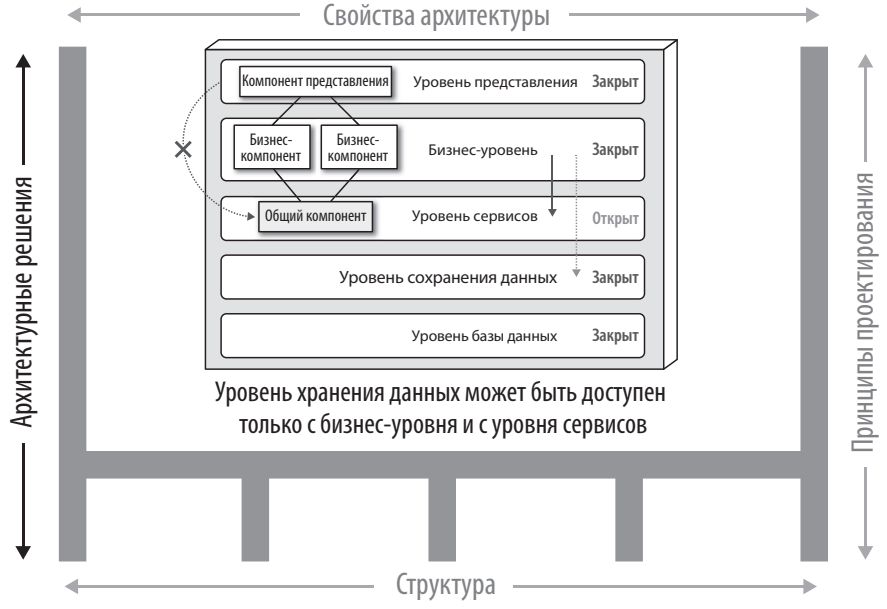

Рис. 1.5. Архитектурные решения являются правилами построения систем

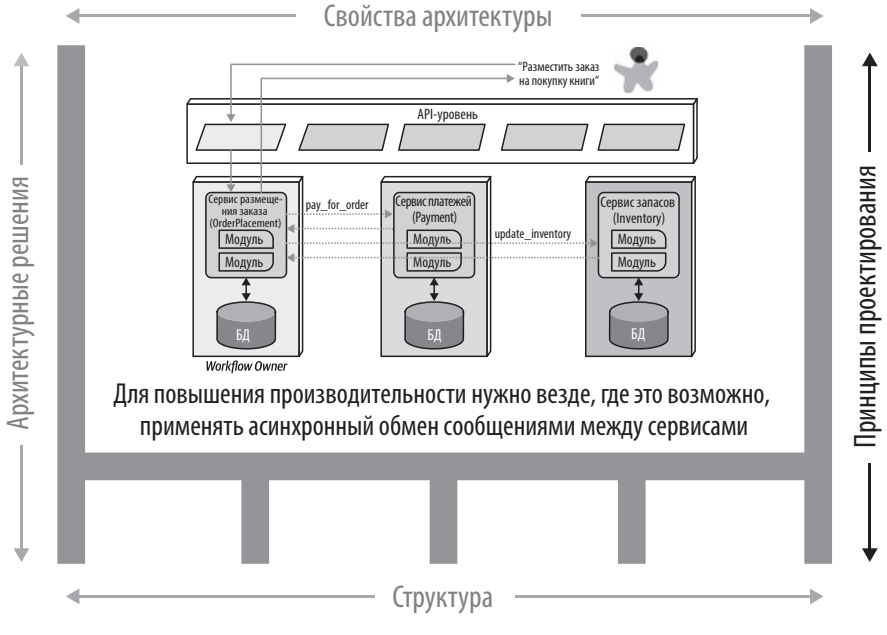

Рис. 1.6. Принципы проектирования являются руководящими установками выстраивания систем

Последним фактором, определяющим архитектуру, являются *принципы проектирования*. Принцип проектирования отличается от архитектурного решения тем, что он является *руководящей установкой*, а не строго определенным правилом. Например, в принципе проектирования, показанном на рис. 1.6, говорится, что для достижения высокой производительности в рамках архитектуры микросервисов команды разработчиков должны использовать асинхронный обмен сообщениями между микросервисами.

Архитектурное решение (правило) никогда не сможет охватить все условия и варианты обмена данными между сервисами, поэтому для обеспечения руководящей установки по предпочтительному методу (в данном случае это асинхронный обмен сообщениями) может использоваться принцип проектирования, позволяющий разработчику выбрать в конкретных обстоятельствах наиболее подходящий протокол обмена данными (например, REST или gRPC).

Ожидания от работы архитектора

Определить роль архитектора ПО ничуть не легче, чем дать определение самой архитектуре. Его обязанности могут варьироваться от задач опытного программиста до специалиста, формирующего стратегическое техническое направление для компании. Так что вместо пустой траты времени на попытку определения этой роли сконцентрируемся на *ожиданиях* от работы архитектора.

Независимо от каких-либо конкретных полномочий, титулов или должностных обязанностей, от архитектора ПО ожидается:

- принятие архитектурных решений;
- постоянный анализ архитектуры;
- своевременное следование последним тенденциям;
- контроль за выполнением принятых решений;
- обладание обширными знаниями и опытом;
- компетентность в нужной области бизнеса;
- владение навыками межличностного общения;
- четкое понимание политики компании.

Эффективная и плодотворная работа в должности архитектора программного обеспечения невозможна без понимания и осуществления каждого из этих ожиданий.

Принятие архитектурных решений

От архитектора ожидаются архитектурные решения и определение принципов проектирования, которые будут служить руководством для принятия технологических решений внутри команды, отдела или всего предприятия.

Ключевым здесь является слово *руководство*. Архитектор должен *выдавать установки*, 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 техническое решение, а надо бы — архитектурное решение: определить принцип проектирования, помогающий команде разработчиков сделать свой выбор. Архитектор должен был бы рекомендовать команде разработчиков *использовать для веб-разработки внешнего интерфейса реактивно-ориентированную среду* и тем самым направить на выбор между Angular, Elm, React.js, Vue или любым другим фреймворком на реактивной основе.

Управлять выбором технологии через архитектурные решения и принципы проектирования довольно сложно. Ключом к принятию эффективных архитектурных решений является вопрос о том, помогает ли архитектурное решение *направлять* команды на выбор правильных технических решений или же такой выбор *делается* за них. И тем не менее бывает так, что для соблюдения конкретных архитектурных свойств, например масштабируемости, производительности или доступности, архитектору приходится принимать те или иные технологические решения. В таком случае это все равно будет считаться архитектурным решением, даже если им определяется конкретная технология. Архитекторы часто испытывают трудности с поиском правильного курса, поэтому архитектурным решениям целиком посвящена глава 19.

Постоянный анализ архитектуры

Архитектор должен постоянно анализировать архитектуру и текущую технологическую среду, а затем 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 технологии. Исходя из нашего опыта, далеко не все архитекторы уделяют достаточно внимания постоянному анализу существующих архитектур. В результате большинство архитектур подвержено структурному распаду, происходящему из-за вноси-

мых разработчиками изменений в код или в проект, что влияет на требуемые свойства архитектуры, например на производительность, доступность или масштабируемость.

Другие аспекты, 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 в плане общей архитектуры.

Чтобы убедиться в состоятельности архитектуры, архитектор должен проводить комплексный анализ изменений в технологиях и в областях решаемых задач. Хотя в объявлениях о вакансии подобное требование к уровню квалификации фигурирует довольно редко, архитектор все же должен соответствовать и этому ожиданию.

Своевременное следование последним тенденциям

Архитектор должен быть в курсе последних тенденций в технологии и в бизнес-секторе.

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 своем бизнес-секторе. Принимаемые архитектором решения носят долговременный характер и трудно поддаются изменениям. Четкое понимание основных тенденций помогает архитектору подготовиться к будущим вызовам и принимать правильные решения.

Отслеживание тенденций и следование им даются нелегко, особенно архитектору ПО. Различные технологии и ресурсы, позволяющие успешно решать эту задачу, будут рассмотрены в главе 24.

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. Рассмотрим такую ситуацию. Архитектор принял решение, что в многоуровневой архитектуре доступ к базе данных возможен только из бизнес- и сервис-уровня (и невозможен с уровня представления). Следовательно, 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 архитектурное решение по конкретной причине: контроль изменений. Закрытие уровней позволяет вносить изменения в базу данных, не затрагивая уровень представления. Если не обеспечить соблюдение архитектурных решений, могут быть допущены такие нарушения, при которых архитектура перестанет отвечать требуемым свойствам (выражаемым словами с окончанием *-ость*), а приложение или система не будут работать так, как ожидалось.

В главе 6 мы подробнее поговорим об оценке соблюдения решений и принципов с помощью автоматизированных функций пригодности и соответствующих автоматизированных инструментов.

Обладание обширными знаниями и опытом

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

Данное требование не означает, что архитектор должен быть великим знатоком каждой среды разработки, платформы и языка, скорее он должен быть знаком с различными технологиями. Большинство современных сред имеет неоднородный характер, и архитектор должен по крайней мере знать возможности взаимодействия нескольких систем и сервисов независимо от языков, платформ и технологий, использованных при написании таких систем или сервисов.

Лучший способ соответствовать этому требованию — это постоянно выходить из зоны комфорта. Сосредоточенность на одной технологии или платформе — это не что иное, как тихая гавань. Успешный архитектор ПО должен активно изучать различные языки программирования, платформы и технологии. Для соответствия данному ожиданию лучше 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 различных средств кэширования и все их «за» и «против», чем быть узким специалистом по одному из них.