ПРОФЕССИОНАЛЬНО

Диомидис Спинеллис
Георгиос Гусиос

ИДЕАЛЬНАЯ АРХИТЕКТУРА

ВЕДУЩИЕ СПЕЦИАЛИСТЫ О КРАСОТЕ ПРОГРАММНЫХ АРХИТЕКТУР

O'REILLY®

СИМВОЛ®

Beautiful Architecture

Leading Thinkers Reveal the Hidden Beauty in Software Design

Diomidis Spinellis and Georgios Gousios

O'REILLY®

ПРОФЕССИОНАЛЬНО

ИДЕАЛЬНАЯ АРХИТЕКТУРА

Ведущие специалисты о красоте программных архитектур

Диомидис Спинеллис и Георгиос Гусиос

СИМВОЛ®

Санкт-Петербург — Москва
2010

Серия «Профессионально»

Диомидис Спинеллис, Георгиос Гусиос

Идеальная архитектура

Ведущие специалисты о красоте программных архитектур

Перевод Е. Матвеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Шалин*
Научный редактор	*А. Брагин*
Редактор	*А. Альбов*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Спинеллис Д., Гусиос Г.

Идеальная архитектура. Ведущие специалисты о красоте программных архитектур. – Пер. с англ. – СПб.: Символ-Плюс, 2010. – 528 с., ил.

ISBN 978-5-93286-175-2

Из каких компонентов строятся надежные, элегантные, гибкие, удобные в сопровождении программные архитектуры? Книга отвечает на этот вопрос серией очерков, написанных ведущими программными архитекторами и проектировщиками современности. В каждом очерке авторы представляют какую-либо выдающую программную архитектуру, анализируют ее отличия от других архитектур и объясняют, почему она идеально подходит для своей цели.

Из книги вы узнаете, как на основе архитектуры Facebook была построена экосистема приложений, ориентированных на работу с данными; как новаторская архитектура Xen повлияла на будущее операционных систем; как процессы в сообществе проекта KDE способствовали превращению программной архитектуры из предварительного проекта в элегантную систему; как «ползучая функциональность» помогла GNU Emacs выйти за пределы изначально запланированных возможностей; как устроена высокооптимизированная виртуальная машина Jikes RVM; какие сходства и различия существуют между объектно-ориентированными и функциональными архитектурными школами; как архитектуры влияют на эволюцию программных продуктов и труд разработчиков.

ISBN 978-5-93286-175-2
ISBN 978-978-0-596-51798-4 (англ)

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Подписано в печать 21.05.2010. Формат 70x90 1/16 . Печать офсетная.
Объем 33 печ. л. Тираж 1500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Оглавление

Предисловие

Стивен Дж. Меллор

Проблемы разработки высокопроизводительных, надежных и качественных программных систем оказались слишком сложными для неформальных, несистематических технических решений, которые работали в прошлом в менее требовательных системах. Сложность наших систем дошла до того уровня, после которого с задачей невозможно справиться без разработки и поддержания общей архитектуры, которая объединяет систему в связное целое и предотвращает разрозненную реализацию со всеми вытекающими проблемами для тестирования и интеграции.

Однако построение архитектуры является весьма сложной задачей. Трудно найти хорошие примеры – иногда из-за закрытого характера решений, иногда, наоборот, из-за необходимости «протолкнуть» некоторый определенный архитектурный стиль в широком диапазоне сред, во многих из которых этот стиль неуместен. К тому же архитектуры весьма масштабны, поэтому их описания часто обескураживают читателей.

И все же в красивых архитектурах проявляются некоторые универсальные принципы, некоторые из которых я кратко опишу ниже:

Один факт в одном месте

Дублирование ведет к ошибкам, поэтому его необходимо избегать. Каждый факт должен быть единым и неделимым и притом независимым от всех остальных фактов. При внесении изменения (а это неизбежно происходит в любой системе) достаточно исправить только одно место. Этот принцип хорошо знаком проектировщикам баз данных, у которых он называется *нормализацией*. В менее формальной степени он также встречается в области проектирования

поведения, когда общая функциональность выделяется в отдельные модули (*факторинг*).

Красивые архитектуры находят способы локализации информации и поведения. Во время выполнения это проявляется в виде деления системы на уровни, каждый из которых представляет некоторый уровень абстракции или предметной области.

Автоматическое распространение

Один факт в одном месте – звучит, конечно, хорошо, но по соображениям эффективности некоторые данные или аспекты поведения часто дублируются. Для сохранения логической целостности и правильности системы распространение этих фактов должно происходить автоматически в ходе построения.

Красивые архитектуры имеют поддержку в виде строительного инструментария, реализующего принципы *метапрограммирования*. Другими словами, один факт в одном месте может распространяться во много мест для повышения эффективности его использования.

Архитектура включает построение

Архитектура должна включать не только систему времени выполнения, но и способ ее построения. Если архитектор уделяет внимание только коду времени выполнения, это верный признак того, что его архитектура постепенно придет в негодность.

Красивые архитектуры метацикличны. Они красивы не только на стадии выполнения, но и на стадии построения, в ходе которого используются те же данные, функции и приемы, что и во время выполнения.

Минимум механизмов

Лучший способ реализации некой функции зависит от конкретного случая, но красивая архитектура не стремится к «самому лучшему». Например, существует множество способов организации хранения данных и поиска в них, но если система может удовлетворить требования производительности, ограничившись использованием одного механизма, разработчику придется писать, проверять и сопровождать меньший объем кода.

Красивые архитектуры используют минимальный набор механизмов, удовлетворяющих общим требованиям. Поиск «самого лучшего» решения для каждого случая ведет к размножению механизмов с повышением риска ошибок, тогда как экономное добавление механизмов позволяет строить более компактные, быстрые и надежные системы.

Построение обобщенных решений

Если вы хотите строить хрупкие, негибкие системы, последуйте совету Айвара Якобсона (Ivar Jacobson) – заложите в основу своей архитектуры сценарии использования и реализацию только одной функции (т. е. используйте объекты-«контроллеры»).

Напротив, расширяемые системы базируются на создании виртуальных машин – обобщенных решений, которые «программируются» данными, предоставляемыми более высокими уровнями, и реализуют сразу несколько функций приложения.

Известно много разных формулировок этого принципа. «Уровни» виртуальных машин восходят еще к Эдгару Дейкстре. «Системы, управляемые данными» предоставляют механизмы, которые базируются на программных инвариантах системы и позволяют данным определить конкретную функциональность для конкретного случая. Они прекрасно подходят для повторного использования – и притом весьма красивы.

Порядок роста

Некогда мы говорили о «порядке» алгоритмов, выражая, скажем, сложность сортировки в виде времени, необходимого для сортировки определенного количества элементов. На эту тему были написаны целые книги.

То же относится и к архитектурным решениям. Например, опрос (polling) лучше всего работает для малого количества элементов, а с увеличением их количества время отклика катастрофически ухудшается. Построение архитектуры на базе прерываний или событий работает хорошо – до того момента, когда они вдруг срабатывают все сразу. Красивые архитектуры учитывают направление возможного роста и принимают соответствующие меры.

Сопротивление энтропии

Красивые архитектуры прокладывают путь наименьшего сопротивления для сопровождения, сохраняющего архитектуру и замедляющего действие закона системной энтропии, который гласит, что с течением времени система становится менее организованной. Программисты, занимающиеся сопровождением, должны хорошо понимать архитектуру, чтобы вносимые изменения соответствовали исходному замыслу, а не увеличивали энтропию системы.

Один из способов основан на применении понятия метафоры из гибких методологий – простого способа описания, «на что похожа» архитектура. Возможны и другие решения, например обширная документация и угрозы увольнения, хотя обычно это действует недолго.

> И все же, как правило, сопровождение основано на качественных инструментах, особенно для генерирования системы. Красивая архитектура должна оставаться красивой.

Эти принципы тесно связаны друг с другом. Принцип «один факт в одном месте» может работать только при наличии автоматического распространения, которое в свою очередь эффективно лишь тогда, когда архитектура учитывает процесс построения. Аналогичным образом конструирование обобщенных решений и минимизация механизмов способствуют соблюдению принципа «один факт в одном месте». Сопротивление энтропии обязательно для сопровождения архитектуры с течением времени, но оно зависит от того, были ли учтены в архитектуре процесс построения и поддержка распространения. Если не учесть направление наиболее вероятного роста системы, архитектура станет нестабильной и в конечном итоге развалится под воздействием экстремальных, но прогнозируемых обстоятельств. А объединение минимальности механизмов с концепцией конструирования обобщенных решений означает, что в красивых архитектурах обычно представлен ограниченный набор шаблонов, позволяющих строить произвольные расширения системы, своего рода «расширение по шаблону».

Короче говоря, красивые архитектуры делают больше меньшими средствами.

Во время чтения этой книги, которую так талантливо собрали и представили Диомидис Спинеллис и Георгиос Гусиос, обращайте внимание на эти принципы и рассматривайте их практическое применение на конкретных примерах, представленных в каждой главе. Также стоит присмотреться к нарушениям принципов и попытаться понять возможные причины – то ли архитектура некрасива, то ли действует какой-то принцип более высокого порядка.

Во время написания предисловия авторы спросили меня, не могу ли я сказать несколько слов о том, как стать хорошим архитектором. Я рассмеялся. Если бы мы знали, как... Но потом я вспомнил по собственному опыту, что существует достаточно мощный (хотя и не аналитический) способ стать хорошим архитектором. Он звучит так[1]: никогда не верьте, что последняя система, которую вы спроектировали, могла быть построена только так, а не иначе, ищите примеры различных решений аналогичных задач. Примеры красивых архитектур, представленные в книге, помогут вам достичь этой цели.

---

[1] Есть и другой вариант: больше тренироваться и меньше есть.

Вступление

Книга, которую вы сейчас читаете, зародилась в 2007 году как продолжение популярной, удостоенной наград книги «Beautiful Code»[1] – подборки статей о новаторских, часто удивительных решениях задач из области программирования. Данная книга отличается от предыдущей и масштабом, и предназначением, но общая идея осталась прежней: ведущие проектировщики и архитекторы описывают выбранную ими программную архитектуру, раскрывают особенности внутреннего строения своих творений и показывают, как они разрабатывают программы функциональные, надежные, удобные, эффективные, удобные в сопровождении и портировании... да, и элегантные.

За материалом для книги мы обратились к ведущим архитекторам хорошо известных или менее известных, но особенно оригинальных программных проектов. Многие из них быстро откликнулись на нашу просьбу и предложили свои идеи, заставляющие читателя задуматься. Некоторые даже застали нас врасплох, предложив вместо статьи о конкретной системе подробно проанализировать глубину и масштаб архитектурных аспектов программирования.

Все авторы были рады узнать, что их усилия в работе над книгой послужат доброму делу, так как авторские отчисления были пожертвованы «Врачам без границ» – международной гуманитарной организации, предоставляющей экстренную медицинскую помощь страдающим людям.

Структура книги

Материал разделен на пять тематических областей: обзоры, корпоративные приложения, системы, пользовательские приложения и язы-

[1] Энди Орам, Грег Уилсон «Идеальный код». – Пер. с англ. – СПб: Питер, 2008.

ки программирования. Бросается в глаза отсутствие глав, посвященных архитектурам программ для настольных систем, – не стоит полагать, что это было сделано преднамеренно. После обращения к более чем 50 проектировщикам программных продуктов такой результат стал для нас полной неожиданностью. Неужели в области программ для настольных систем нет действительно блестящих примеров красивых архитектур? Или талантливые архитекторы стараются держаться подальше от области, в которой развитие часто сводится к тому, что на приложение постоянно навешиваются все новые функции? Нам было бы интересно узнать ваше мнение по этому поводу.

Часть I «Об архитектуре»

В первой части книги изучается широта и масштаб программных архитектур, а также их значение для разработки и эволюции программ.

В главе 1 «Что такое архитектура?», написанной Джоном Клейном и Дэвидом Вайссом, программная архитектура определяется с точки зрения проблем качества и архитектурных структур.

Глава 2 «Повесть о двух системах: сказка для современных программистов», написанная Питом Гудлифом, в аллегорической форме показывает, как программные архитектуры могут влиять на эволюцию системы и на участие разработчиков в проектах.

Часть II «Архитектура корпоративных приложений»

Корпоративные системы – основа работы компьютерных служб многих организаций – часто представляют собой сделанные под заказ конгломераты, обычно построенные из разнообразных компонентов. Они обслуживают большие транзакционные нагрузки, а значит, должны масштабироваться вместе с потребностями предприятия, приспосабливаясь к изменяющимся экономическим реалиям. Важнейшие факторы при проектировании архитектуры таких систем – масштабируемость, корректность, стабильность и расширяемость. Во второй части книги представлены некоторые показательные примеры корпоративных программных архитектур.

Глава 3 «Масштабирование», написанная Джимом Уолдо, демонстрирует архитектурное мастерство, необходимое для построения серверов, обслуживающих массовые многопользовательские сетевые игры.

Глава 4 «Фото на память», написанная Майклом Найгардом, рассматривает архитектуру многоэтапной, многоузловой системы обработки данных, а также описывает компромиссы, необходимые для ее успешной работы.

Глава 5 «Ресурсно-ориентированные архитектуры, жизнь в WWW», написанная Брайаном Слеттеном, показывает возможности привязки к ресурсам при конструировании приложений, управляемых данными, и дает элегантный пример чистой ресурсно-ориентированной архитектуры.

Глава 6 «Архитектура платформы Facebook», написанная Дэйвом Феттерманом, разъясняет достоинства систем, в которых центральное место занимают данные, а также объясняет, как хорошая архитектура создает и поддерживает экосистему приложения.

Часть III «Системная архитектура»

Системные программы по праву считаются самыми сложными для проектирования – отчасти из-за того, что эффективная работа с оборудованием остается волшебством, доступным лишь немногим избранным, а отчасти из-за того, что многие относятся к системным программам как к инфраструктуре, которая «просто находится на своем месте». Великие системные архитектуры редко строятся на пустом месте; большинство современных систем базируется на идеях, впервые высказанных в 1960-е годы. В главах части III читатель познакомится с четырьмя новаторскими программными архитектурами, а также узнает, какие сложности, лежащие за архитектурными решениями, сделали их такими красивыми.

Глава 7 «Xen и красота виртуализации», написанная Дереком Мюрреем и Кайром Фрейзером, дает пример того, как хорошо продуманная архитектура изменяет пути эволюции операционных систем.

Глава 8 «Guardian: отказоустойчивая операционная система», написанная Грегом Лехи, содержит ретроспективный анализ архитектурных решений и структурных элементов (как программных, так и аппаратных), благодаря которым платформа Tandem почти 20 лет оставалась лидером в области сред высокой доступности.

Глава 9 «JPC: эмулятор x86 PC на языке Java», написанная Ризом Ньюманом и Кристофером Деннисом, описывает, как тщательное проектирование и хорошее понимание требований предметной области помогают преодолеть кажущиеся недостатки программной системы.

Глава 10 «Метациклические виртуальные машины: Jikes RVM», написанная Иэном Роджерсом и Дэйсом Гроувом, проведет читателя по пути принятия архитектурных решений, необходимых для создания самооптимизируемой, самодостаточной среды времени выполнения для высокоуровневого языка.

Часть IV «Архитектуры пользовательских приложений»

Пользовательские приложения – те программы, с которыми мы больше всего общаемся в своей повседневной жизни, и они же создают основную нагрузку на процессоры наших компьютеров. Такие программы обычно не нуждаются в тщательном управлении ресурсами или обслуживании больших объемов транзакций. Тем не менее пользовательские приложения должны быть удобными, безопасными, настраиваемыми и расширяемыми. Эти свойства делают их популярными и широко распространенными, а в случае бесплатных программ и программ с открытым кодом порождают армию добровольцев, желающих потрудиться над их совершенствованием. В части IV авторы анализируют архитектуры и процессы в сообществах, необходимые для эволюции двух очень популярных программных пакетов.

Глава 11 «GNU Emacs: сила ползучей функциональности», написанная Джимом Блэнди, объясняет, как набор очень простых компонентов и язык расширения превращают скромный текстовый редактор в ~~операционную систему~~[1] универсальный инструмент в рабочем арсенале программиста.

Глава 12 «Когда базар строит собор», написанная Тилем Адамом и Мирко Бёмом, показывает, как процессы в сообществах (такие как рабочие встречи и равноправный анализ кода) превращают программные архитектуры из предварительных набросков в прекрасные системы.

Часть V «Языки и архитектура»

Как сообщается во многих книгах по программированию, язык, который мы используем, влияет на способ решения задачи. Но может ли язык программирования также повлиять на архитектуру системы, и если да, то как? В строительной архитектуре новые материалы и распространение систем автоматизированного проектирования позволили выражать более сложные, а иногда невероятно красивые планы. Существует ли нечто аналогичное в области компьютерных программ? В части V, состоящей из двух последних глав, рассматриваются отношения между используемыми инструментами и создаваемыми архитектурами.

Глава 13 «Программные архитектуры: объектно-ориентированные и функциональные», написанная Бертраном Мейером, сравнивает пре-

[1] Как говорят некоторые фанатичные пользователи, «Emacs – моя операционная система, Linux просто предоставляет драйверы устройств».

имущества двух архитектурных стилей, объектно-ориентированного и функционального.

Глава 14 «Перечитывая классику», написанная Панайотисом Лурида-сом, содержит обзор архитектурных решений, заложенных в основу современных и классических объектно-ориентированных языков.

Наконец, в своем замечательном послесловии Уильям Дж. Митчелл, профессор архитектуры, мультимедийного искусства и науки Массачу-сетского технологического института, связывает концепции красоты зданий, возводимых в реальном мире, и программных архитектур, существующих только виртуально.

Принципы, свойства и структуры

На поздней стадии подготовки материалов один из рецензентов предложил высказать наше личное мнение, что читатель должен узнать из каждой главы. Идея была заманчивой, но нам не захотелось угадывать намерения авторов. Обратиться к самим авторам за метаанализом их материалов? Это кончилось бы возведением Вавилонской башни из определений, терминов и архитектурных конструкций, которые наверняка собьют с толку читателя. Нам была нужна общая номенклатура архитектурных терминов; к счастью, мы вовремя поняли, что она у нас уже есть.

В своем предисловии Стивен Мэллор обсуждает семь принципов, на которых базируются все красивые архитектуры. В главе 1 Джон Клейн и Дэвид Вайсс описывают четыре основных структурных элемента архитектур и шесть свойств, присущих красивым архитектурам. Внимательный читатель заметит, что принципы Мэллора и свойства Вайсса и Клейна частично взаимозависимы. Более того, они в основном совпадают; это происходит из-за того, что великие умы думают в похожих направлениях. Трое авторов – очень опытные архитекторы, и они неоднократно убеждались в важности описываемых ими принципов на собственном опыте.

Мы объединили архитектурные принципы Мэллора с определениями Клейна и Вайсса в двух списках: в первом перечислены принципы и свойства (табл. В.1), а во втором – структурные элементы (табл. В.2). Затем мы попросили авторов каждой статьи отметить те пункты, которые, по их мнению, относились к их материалу, и снабдили каждую главу соответствующим пояснением. В таблицах приведены определения всех принципов, свойств и архитектурных концепций, присутствующих в начале каждой главы. Надеемся, эти пояснения, дающие четкий обзор содержимого каждой главы, помогут вам ориентироваться в книге.

Таблица В.1. Архитектурные принципы и свойства

Принцип или свойство	Означает, что архитектура...
Гибкость	Предоставляет «подходящие» механизмы для решения разнообразных задач с относительно небольшим объемом выразительных средств.
Концептуальная целостность	Предоставляет один оптимальный, лишенный избыточности способ выражения решения группы сходных задач.
Возможность независимого изменения	Имеет изолированные элементы, что сводит к минимуму количество мест внесения изменений при модификации.
Автоматическое распространение	Поддерживает логическую целостность и правильность работы посредством распространения изменений данных или поведения между модулями.
Удобство построения	Управляет правильным и логичным процессом построения программного продукта.
Адаптация к росту	Может приспособиться к возможному росту.
Сопротивление энтропии	Поддерживает порядок за счет принятия, ограничения и изоляции последствий изменений.

Таблица В.2. Архитектурные структуры

Структура	Назначение
Модуль	Скрывает решения проектирования или реализации за стабильным интерфейсом.
Зависимость	Упорядочивает компоненты в соответствии с использованием той или иной функциональности.
Процесс	Инкапсулирует и изолирует состояние модуля на стадии выполнения.
Доступ к данным	Разбивает данные на категории с назначением прав доступа к ним.

Условные обозначения

В книге используются следующие условные обозначения:

Курсив

Новые термины, URL-адреса, адреса электронной почты, имена и расширения файлов.

`Моноширинный шрифт`

Используется в листингах программ, а также в тексте для обозначения программных элементов (имена переменных и функций, баз данных, типов данных, переменных среды, команд и ключевых слов).

`Моноширинный полужирный шрифт`

Команды или другой текст, который должен вводиться в показанном виде.

`Моноширинный курсив`

Текст, который должен заменяться пользовательскими значениями (или значениями, определяемыми по контексту).

Использование примеров кода

Эта книга призвана помочь вам в решении конкретных задач. В общем случае вы можете использовать приводимые примеры кода в своих программах и документации. Связываться с авторами для получения разрешения не нужно, если только вы не воспроизводите значительный объем кода. Например, если ваша программа использует несколько фрагментов кода из книги, обращаться за разрешением не нужно. С другой стороны, для продажи или распространения дисков CD-ROM с примерами из книг O'Reilly потребуется разрешение. Если вы отвечаете на вопрос на форуме, приводя цитату из книги с примерами кода, обращаться за разрешением не нужно. Если значительный объем кода из примеров книги включается в документацию по вашему продукту, разрешение необходимо.

Мы будем признательны за ссылку на источник информации, хотя и не требуем ее. Обычно в ссылке указывается название, автор, издательство и код ISBN, например: «*Beautiful Architecture*, edited by Diomidis Spinellis and Georgios Gousios. Copyright 2009 O'Reilly Media, Inc., 978-0-596-51798-4».

Если вы полагаете, что ваши потребности выходят за рамки оправданного использования примеров кода или разрешений, приведенных выше, свяжитесь с нами по адресу *permissions@oreilly.com*.

Safari® Enabled

Если на обложке вашей любимой технической книги есть пиктограмма Safari® Enabled, это означает, что книга доступна через O'Reilly Network Safari Bookshelf.

Система Safari лучше обычных электронных книг. Это целая виртуальная библиотека с возможностью поиска по тысячам лучших технических книг, копирования/вставки примеров кода, загрузки глав и быстрого поиска ответов, когда вам потребуется самая точная и актуальная информация. Safari можно бесплатно опробовать на сайте *http://safari.oreilly.com*.

Как с нами связаться

Пожалуйста, со всеми комментариями и вопросами по книге обращайтесь к издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (США или Канада)
707-829-0515 (международные или местные звонки)
707 829-0104 (факс)

На сайте издательства имеется веб-страница книги со списками обнаруженных опечаток и другой дополнительной информацией. Страница доступна по адресу:

http://www.oreilly.com/catalog/9780596517984

С комментариями и техническими вопросами по поводу книги обращайтесь по электронной почте:

bookquestions@oreilly.com

За дополнительной информацией о книгах, конференциях, ресурсах и O'Reilly Network обращайтесь на сайт O'Reilly:

http://www.oreilly.com

Благодарности

Издание книги является делом коллективным, особенно если эта книга – сборник статей. Мы благодарны многим людям. Прежде всего, мы благодарим соавторов, которые своевременно предоставили материал, а потом внимательно относились к нашим просьбам по поводу различных изменений и дополнений. Рецензенты книги, Роберт А. Максимчук (Robert A. Maksimchuk), Гэри Поллис (Gary Pollice), Дэвид Уэст (David West), Грег Уилсон (Greg Wilson) и Бобби Янг (Bobbi Young), поделились полезными замечаниями по поводу улучшения каждой главы и книги в целом. Наш редактор в издательстве O'Reilly Мэри Треслер (Mary Treseler) помогла нам с поиском авторов, организовала

процесс рецензирования и управляла ходом работы с потрясающей эффективностью. Позднее Сара Шнейдер (Sarah Schneider) работала с нами в качестве редактора по производству, в сжатые сроки умело справляясь с нередко противоречивыми требованиями. Выпускающий редактор Женевьева д'Антремон (Genevieve d'Entremont) и составитель алфавитного указателя Фред Браун (Fred Brown) мастерски сформировали из материала, собранного со всего мира, книгу, которая читается легко, словно написанная одним пером. Художник Роберт Романо (Robert Romano) преобразовал графику в разных форматах (включая наброски, сделанные от руки) в профессиональные диаграммы, которые вы видите в книге. Дизайнер обложки Карен Монтгомери (Karen Montgomery) создала красивую и привлекательную обложку, соответствующую содержанию книги, а художник-оформитель Дэвид Футато (David Futato) предложил оригинальную и удобную схему интеграции сводки пояснений в дизайн книги. Наконец, мы хотим поблагодарить за поддержку свои семьи и друзей, ведь мы уделяли этой книге то внимание, которое по праву принадлежало им.

I

Об архитектуре

1

Что такое архитектура?

Джон Клейн
Дэвид Вайсс

Введение

Термин «архитектура» используют строители, музыканты, писатели, проектировщики компьютеров и сетей, программисты (и не только они; когда-нибудь слышали о «кулинарных архитекторах»?) – но с разными результатами. Здание совершенно не похоже на симфонию, но у того и другого есть архитектура. Кроме того, все архитекторы непременно упоминают о красоте своей работы и ее результатов. Строитель скажет, что здание должно создавать хорошие условия для жизни или работы и что оно должно быть красивым. Музыкант скажет, что музыка должна быть хорошо исполнена, в ней должна четко прослеживаться тема и она должна быть красивой. Программный архитектор скажет, что система должна быть дружественной к пользователю и быстро реагировать на его действия, быть простой в установке и сопровождении, надежной и свободной от критических ошибок; кроме того, система должна стандартно взаимодействовать с другими системами и – тоже должна быть красивой.

В этой книге приводятся подробно описанные примеры красивых архитектур из области компьютеризированных систем. Поскольку наша

дисциплина относительно молода, у нас меньше образцов для подражания, чем в таких областях, как строительство, музыка или литература, а потому потребность в них еще сильнее. Книга призвана заполнить этот пробел.

Прежде чем переходить к конкретным примерам, нам хотелось бы разобраться, что же такое архитектура вообще и какими признаками обладает красивая архитектура. Как видно из разных определений архитектуры, приводимых в этой главе, в каждой области существует собственное определение, поэтому мы сначала исследуем общие признаки архитектур из разных областей и проблем, решаемых с их помощью. В частности, архитектура помогает нам обеспечить соответствие системы потребностям заинтересованных сторон, а также справиться со сложностями планирования, построения и сопровождения системы.

Далее мы переходим к определению архитектуры. Вы увидите, как это определение применяется к архитектуре программных продуктов, поскольку именно они занимают центральное место во многих последующих примерах. Ключевым моментом определения является то, что архитектура состоит из набора структур, спроектированных специально для того, чтобы архитекторы, строители и другие заинтересованные стороны могли видеть, в какой степени система удовлетворяет их потребностям.

Глава завершается обсуждением признаков красивых архитектур; мы рассмотрим несколько примеров. В красоте центральное место занимает концептуальная целостность – набор абстракций и правил их по возможности простого использования в системе.

В нашем обсуждении термином «архитектура» будет обозначаться совокупность артефактов (включая документацию: планы, строительные спецификации и т. д.), описывающих создаваемый объект, при этом сам объект рассматривается как набор структур. Некоторые авторы также используют термин для описания процесса создания артефактов, включая конечный результат работы. Но, как указывают Джим Уолдо (Jim Waldo) и другие, не существует методологии, которая гарантировала бы создание хорошей архитектуры, не говоря уже о красивой (Waldo 2006), поэтому наше внимание будет в большей степени сосредоточено на артефактах, а не на процессах.

Архитектура: «Искусство или наука строительства; прежде всего, искусство и практика проектирования и строительства зданий, предназначенных для использования человеком, с учетом эстетических и практических факторов».

Краткий Оксфордский словарь английского языка, 5-е издание, 2002

Во всех отраслях знаний архитектура предоставляет средства для решения стандартной задачи: обеспечения того, что построенное здание, мост, музыкальное сочинение, книга, компьютер, сеть или система будут обладать определенными свойствами или функциями. Иначе говоря, архитектура является как планом, который гарантирует наличие у системы желаемых свойств, так и описанием построенной системы. В Википедии говорится: «Согласно самой ранней из дошедших до нас работ по теме – трактату Витрувия «Об архитектуре» – хорошее здание должно обладать красотой (Venustas), прочностью (Firmitas) и практичностью (Utilitas); архитектура может быть определена как баланс и сочетание этих трех элементов, при котором ни один не подчиняет себе остальные».

Мы говорим об «архитектуре» симфонии,
и называем архитектуру «застывшей музыкой».

Дерик Кук «The Language of Music»

Хорошей архитектуре системы присуща концептуальная целостность; другими словами, она дополняется набором правил проектирования, которые способствуют снижению сложности системы и могут использоваться как руководство по ее проектированию и проверке. Правила проектирования могут включать некоторые абстракции, которые всегда используются одинаково, например виртуальные устройства. Правила также могут быть представлены в виде моделей, или шаблонов, скажем каналов и фильтров. В лучшем случае архитектура включает правила, поддающиеся объективной проверке, например: «любое виртуальное устройство определенного типа может заменить любое другое виртуальное устройство того же типа в случае его отказа» или «все процессы, конкурирующие за один ресурс, должны обладать одинаковым приоритетом планирования».

Современный архитектор может сказать, что конструируемый объект или система должны обладать следующими характеристиками:

- Функциональность, необходимая заказчику
- Возможность построения в заданный срок
- Адекватность функционирования
- Надежность
- Удобство использования
- Безопасность
- Приемлемость по затратам
- Соответствие законодательству
- Превосходство в сравнении с предшественниками и конкурентами

Архитектура компьютерной системы – это минимальный набор свойств, определяющих, какие программы будут работать в системе и какие результаты они дадут.

Джеррит Блау и Фредерик Брукс «Computer Architecture»

Мы еще не видели ни одной сложной системы, которая бы идеально соответствовала всем перечисленным характеристикам. Архитектура является искусством компромисса – решение, усиливающее одну из этих характеристик, часто ослабляет другие. Архитектор должен определить, в какой степени достаточно удовлетворять те или иные требования, для чего он выявляет важные запросы конкретной системы и приемлемые уровни их выполнения.

Одно из основных понятий архитектуры – концепция структур, определяемых различными компонентами и их отношениями: согласованиями, коммуникациями, синхронизациями и другими взаимодействиями. Такими компонентами могут быть опорные балки или внутренние помещения в здании, отдельные партии музыкальных инструментов или мелодии в симфонии, главы или персонажи книги, процессоры и микросхемы памяти в компьютере, уровни стека коммуникационных протоколов или процессоры, подключенные к сети, взаимодействующие последовательные процессы, объекты, коллекции макросов времени компиляции или сценарии времени выполнения. В каждой отрасли знаний существуют свои виды компонентов и отношений между ними.

В более широком понимании термин «архитектура» всегда означает «неизменная глубинная структура».

Стюарт Бранд «How Buildings Learn»

Несмотря на все возрастающую сложность систем и их взаимодействий (как внутренних, так и внешних), архитектура, складывающаяся из набора структур, предоставляет основные средства для снижения уровня сложности- и обеспечения необходимых свойств у итоговой системы. Структуры дают возможность интерпретировать систему как набор взаимодействующих компонентов.

Каждая структура призвана помочь архитектору понять, каким образом удовлетворить ту или иную потребность, например способность к изменениям или производительность системы. Возможно, доказывать, что конкретная потребность действительно удовлетворяется, придется кому-то другому, но архитектор должен быть способен продемонстрировать, что *все* потребности были учтены при проектировании.

Сетевая архитектура: коммуникационное оборудование, протоколы и каналы передачи данных, образующие сеть, и методы их организации.

http://www.wtcs.org/snmp4tpc/jton.htm

Роль архитектора

При проектировании, строительстве или реконструкции зданий ведущие проектировщики назначаются «архитекторами», и на них возлагается широкий круг обязанностей. Архитектор готовит исходные эскизы, на которых обозначены внешний вид здания и внутренняя структура помещений, затем обсуждает их с клиентами, пока все заинтересованные стороны не согласятся с тем, что изображено именно то, что им нужно. Эскизы являются абстракциями: основное внимание на них уделяется наиболее существенным подробностям конкретного аспекта здания, а другие аспекты игнорируются.

После того как архитектор согласует эти абстракции со своими клиентами, он готовит гораздо более подробные чертежи (или наблюдает за их подготовкой), а также соответствующие текстовые спецификации. Эти чертежи со спецификациями описывают многие «житейские подробности» здания: трубопроводы, материал наружной отделки, остекление окон и электропроводку.

Изредка архитектор просто передает подробные чертежи строителю, который возводит по ним здание. В более важных проектах архитектор продолжает участвовать в строительстве, периодически проверяет работу, а также может предлагать изменения или принимать пожелания как от строителя, так и от клиента. Если архитектор наблюдает за ходом проекта, то последний считается законченным лишь после того, как архитектор подтвердит, что построенное здание в основном соответствует всем чертежам и спецификациям.

Мы привлекаем к работе архитектора, чтобы убедиться в том, что проект: (1) удовлетворяет потребностям клиента, включая упоминавшиеся ранее характеристики; (2) обладает концептуальной целостностью, то есть во всех аспектах проекта используются одни и те же правила проектирования; (3) удовлетворяет существующим законам и требованиям безопасности. Одна из важных обязанностей архитектора заключается в наблюдении за последовательным применением концепций проектирования в ходе реализации проекта.

Иногда архитектор также выполняет функции посредника между строителем и клиентом. По поводу того, какие решения находятся в ведении архитектора, а какие принимаются другими людьми, часто возникают разногласия, но всегда ясно, что архитектор принимает все критически важные решения, в том числе те, которые могут повлиять

на удобство использования, безопасность и эксплуатационную надежность конструкции.

Сочинение музыки и программные архитектуры

Хотя разработку программных архитектур часто сравнивают со строительством зданий, возможно, сочинение музыки является более правильной аналогией. Архитектор-строитель создает статическое описание (чертежи и другие рисунки) относительно статичной конструкции (относительно – потому что архитектура должна учитывать движение людей и вспомогательных механизмов в здании, а также распределение нагрузок). При сочинении музыки (и проектировании программных продуктов) композитор (архитектор программного продукта) создает статическое описание музыкального произведения (описание архитектуры и кода), которое позднее многократно исполняется (запускается). И в музыке, и в программировании архитектура может учитывать взаимодействие многочисленных компонентов для получения желаемого результата, а сам результат зависит от исполнителя, среды исполнения и интерпретации исходного замысла исполнителем.

Роль программного архитектора

В проектах по разработке программного обеспечения необходимы люди, которые играют в построении продукта ту же роль, что и традиционные архитекторы при строительстве или реконструкции зданий. Но в программных системах никогда не существовало полной ясности по поводу того, какие решения являются прерогативой архитектора, а какие можно оставить людям, занимающимся реализацией. Сложности с определением функций архитектора в программном проекте обусловлены тремя факторами: отсутствием традиций, нематериальной природой продукта и сложностью системы. (Панорама выполнения архитектором своих функций в крупной организации, занимающейся разработкой ПО, представлена у Гринтера [Grinter 1999].)

В частности:

- Архитектор-строитель может обратиться к тысячелетней истории и посмотреть, как работали архитекторы прошлого. Он может посещать и изучать здания, которые стоят сотни, а иногда и более тыся-

чи лет, и все еще используются. История программной отрасли насчитывает всего несколько десятков лет, а наши разработки часто не предназначены для публики. Кроме того, у архитекторов-строителей давно существуют стандарты для описания чертежей и спецификаций, благодаря чему современные архитекторы могут пользоваться документированной историей архитектуры.

- Здания являются материальными продуктами; существуют принципиальные различия между планами, созданными архитектором, и зданием, построенным рабочими.

В крупных программных проектах часто участвует много архитекторов. Некоторые архитекторы обладают узкой специализацией (скажем, базы данных или сети) и обычно работают в составе групп, но в тексте книги мы будем предполагать, что архитектор только один.

Из чего складывается программная архитектура?

Было бы неправильно относиться к архитектуре как к простой сущности, которая может быть описана одним документом или рисунком. Архитектору приходится принимать много решений. Чтобы эти решения приносили пользу, их необходимо документировать для последующего анализа, обсуждения, изменения и утверждения, после чего они используются для принятия решений и конструирования продукта. В программных системах у этих проектировочных решений имеются поведенческие и структурные аспекты.

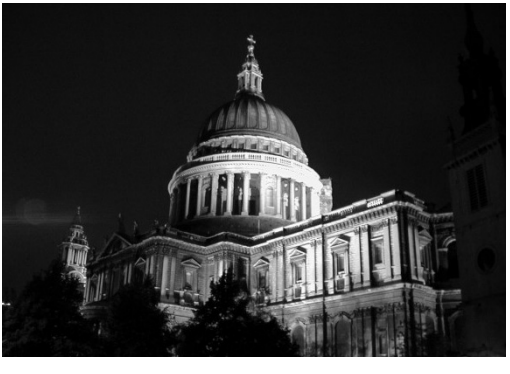
Внешние поведенческие описания показывают, как продукт будет взаимодействовать со своими пользователями, другими системами и внешними устройствами. Эти описания должны иметь форму требований. Структурные описания показывают, как продукт делится на части и как эти части связаны друг с другом. Внутренние поведенческие описания представляют интерфейсы между компонентами. В структурных описаниях часто демонстрируются несколько альтернативных представлений одной части, потому что всю информацию невозможно содержательно изложить в одном рисунке или документе. Компонент в одном представлении может быть составной частью компонента в другом представлении.

Программные архитектуры часто представляются в виде многоуровневых иерархий, склонных к смешению нескольких разных структур на одной диаграмме. В 1970-е годы Парнас указал, что термин «иерархия» стал модным словечком, которое часто применяется некорректно, затем дал точное определение термина и привел несколько примеров структур, используемых в разных целях при проектировании разных систем (Parnas 1974). Описание архитектурных структур в виде

Повторное использование архитектуры

Для поддержания огромного купола Софийского собора (верхняя иллюстрация), построенного в Константинополе (ныне Стамбул) в VI веке, были впервые применены конструкции, называемые «парусами». Собор считается одним из шедевров византийской архитектуры.

Спустя 1100 лет Кристофер Рен использовал аналогичное решение при строительстве купола собора Святого Павла (нижняя иллюстрация), архитектурной достопримечательности Лондона. Оба здания до сих пор стоят и продолжают использоваться в наши дни.



набора *представлений* (*views*), ориентированных на решение разных задач, в настоящее время считается стандартной практикой в архитектуре (Clements et al. 2003; IEEE 2000). Мы будем использовать термин «архитектура» для обозначения набора прокомментированных диаграмм и функциональных описаний, которые определяют структуры, используемые для проектирования и конструирования систем. В сообществе разработчиков ПО существует много используемых и рекомендуемых форм таких диаграмм и описаний. Некоторые примеры приведены у Хоффмана и Вайсса (Hoffman and Weiss, 2000, главы 14 и 16).

Архитектура программы или компьютерной системы представляет собой совокупность структур системы, состоящих из программных элементов, внешне видимых свойств этих элементов и отношений между ними.

«Внешне видимыми свойствами» называются ожидания других элементов в отношении данного элемента: предоставляемый сервис, характеристики быстродействия, механизм обработки ошибок, использование общих ресурсов и т. д.

Лен Басс, Пол Клементс и Рик Казман,
«Software Architecture in Practice», Second Edition

Архитектура и композиция системы

Архитектура является частью композиции системы; она выводит на первый план некоторые подробности, абстрагируясь от других. Таким образом, архитектура является подмножеством композиции. Разработчик, все внимание которого сосредоточено на реализации компонента системы, может недостаточно хорошо представлять себе схему взаимодействия всех компонентов; его интересует только композиция и разработка небольшого числа компонентов, в том числе архитектурные ограничения, которые должны соблюдаться, и правила, которые они могут использовать. Следовательно, разработчик имеет дело с другим аспектом композиции системы, нежели архитектор.

Если архитектура ориентируется на отношения между компонентами и внешне видимые свойства системных компонентов, то композиция помимо этого ориентируется на внутреннюю структуру этих компонентов. Например, если один набор компонентов состоит из модулей, скрывающих информацию, то внешне видимые свойства образуют интерфейсы этих компонентов, а к внутренней структуре относятся структуры данных и передача управления внутри модуля (Hoffman and Weiss 2000, главы 7 и 16).

Создание программной архитектуры

До настоящего момента мы рассматривали «архитектуру вообще», а также исследовали сходство и различия программных архитектур в сравнении с архитектурами из других областей. Пришло время перейти от вопроса «что?» к следующему вопросу – «как?» На чем должен сосредоточить свое внимание архитектор при создании архитектуры программной системы?

Функциональность системы не является основной заботой программного архитектора.

Да, вы не ошиблись – функциональность системы не является основной заботой программного архитектора.

Допустим, вас наняли для разработки архитектуры «веб-приложения». Начнете ли вы расспрашивать о макете страниц и деревьях навигации или же зададите следующие вопросы:

- Кто предоставит хостинг? Существуют ли в среде хостер-провайдера ограничения на применяемые технологии?
- Будет ли приложение работать в Windows Server, или в стеке LAMP?
- Сколько одновременно работающих пользователей должно поддерживать приложение?
- Насколько безопасным должно быть приложение? Существуют ли данные, которые необходимо защитить? Будет ли приложение развернуто для общего доступа в Интернете или в приватной интрасети?
- Можно ли назначить приоритеты ответам на эти вопросы? Например, можно ли считать, что количество пользователей важнее времени отклика?

В зависимости от ответов на эти и некоторые другие вопросы можно переходить к построению эскиза архитектуры системы. А ведь мы еще ни слова не сказали о функциональности приложения.

Допустим, в данном случае мы немного слукавили – область веб-приложений достаточно хорошо изучена, поэтому вы уже знали, какие решения окажут наиболее заметное влияние на архитектуру. Аналогичным образом, если бы речь зашла о телекоммуникационной системе или авиационной электронике, архитектор с опытом работы в соответствующей области уже имел бы некоторое представление о необходимой функциональности. Но, несмотря на это, факт остается фактом: вы смогли приступить к созданию системы, не беспокоясь о ее функциональности. Внимание было направлено на *качественные требования*, которые необходимо было удовлетворить.

Качественные требования определяют, каким образом должна быть предоставлена функциональность системы, чтобы она оказалась приемлемой для сторон, финансово заинтересованных в успехе системы. У этих сторон имеются определенные потребности, о которых архитектор должен позаботиться. Позднее мы обсудим проблемы, которые часто возникают, когда необходимо обеспечить наличие у системы обязательных качеств. Как говорилось ранее, архитектор среди прочего должен обеспечить соответствие системы потребностям клиента, и для упрощения понимания этих потребностей используются качественные требования.

Этот пример выделяет два важных принципа, типичных для успешных архитекторов: привлечение к работе заинтересованных сторон и ориентация как на качественные требования, так и на функциональность. Как архитектор вы должны сначала спросить у клиента, чего он хочет от системы и каковы приоритеты его потребностей. В реальном проекте также следовало бы обратиться к другим заинтересованным сторонам. Типичные заинтересованные стороны и их цели:

- Финансисты, которые желают знать, может ли проект быть завершен в пределах отведенного времени и ресурсов.
- Архитекторы, разработчики и тестеры, для которых сначала важна фаза исходного строительства, а позднее – фазы сопровождения и развития.
- Руководители проектов, которые должны организовывать группы и планировать этапы.
- Специалисты по маркетингу, использующие качественные требования для описания отличий системы от разработок конкурентов.
- Пользователи: конечные пользователи, системные администраторы и люди, занимающиеся установкой, развертыванием, техническим обеспечением и настройкой системы.
- Персонал технической поддержки, для которого важно количество и сложность обращений в их службу.

У каждой системы существуют свои качественные требования. Некоторые из них (такие как производительность, безопасность и масштабируемость) могут иметь четкое формальное определение, но другие, часто не менее важные (например, способность к изменениям, простота сопровождения и удобство использования), невозможно определить достаточно четко, чтобы от этого была хоть какая-нибудь польза. Не правда ли, странно: заинтересованные стороны желают реализовывать функции в программном коде, а не в оборудовании, чтобы при необходимости их можно было легко и быстро изменить, а потом едва упоминают возможность внесения изменений в формулировках каче-

ственных требований? От архитектурных решений зависит то, какие изменения будут вноситься легко и быстро, а какие потребуют времени и значительных затрат. Так разве не должен архитектор понимать ожидания своих клиентов в отношении таких качеств, как «способность к изменениям», так же хорошо, как он понимает функциональные требования?

Итак, архитектор разобрался в качественных требованиях заинтересованных сторон проекта. Что он должен делать после этого? Проанализировать компромиссы. Например, шифрование сообщений повышает безопасность, но ухудшает производительность. Конфигурационные файлы улучшают способность к изменениям, но могут снизить удобство пользования (если мы не сможем организовать проверку правильности конфигурации). Будем ли мы использовать стандартный формат таких файлов (например, XML), или изобретем свое собственное представление? В ходе проектирования архитектуры системы приходится принимать много непростых компромиссных решений.

Следовательно, работа архитектора должна начаться с общения с заинтересованными сторонами. В ходе этого общения архитектор выясняет качественные требования и ограничения, назначая им приоритеты. Почему бы не начать с функциональных требований? У любой системы обычно существует много возможных декомпозиций. Например, если выбрать в качестве отправной точки модель данных, вы придете к одной архитектуре; если начать с модели бизнес-процессов, архитектура может быть совершенно другой. В крайнем случае декомпозиция вообще отсутствует, а система разрабатывается как монолитный блок программного кода. Такой подход обеспечит выполнение всех функциональных требований, но, скорее всего, в этом случае не будут выполняться качественные требования – способность к изменениям, удобство сопровождения и масштабируемость.

Архитекторам часто приходится перерабатывать системы на архитектурном уровне – например, для перехода от простой схемы развертывания к распределенной, или от однопоточной модели к многопоточной (для удовлетворения требований к масштабируемости или производительности), или от жестко закодированных параметров к внешним конфигурационным файлам (потому что параметры, которые *никогда* не должны были изменяться, теперь вдруг изменились).

Обычно архитекторы умеют удовлетворять функциональные требования, но лишь немногие из них способны также позаботиться об удовлетворении качественных требований. Давайте вернемся к примеру с веб-приложением. Подумайте, сколько существует разных способов построения веб-страниц – Apache со статическими страницами, CGI, серв-

леты, JSP, JSF, PHP, Ruby on Rails, ASP.NET и т. д. Выбор одной из этих технологий как архитектурного решения значительно повлияет на возможность выполнения качественных требований. Например, выбор Ruby on Rails обеспечит быстрый выход на рынок, но может затруднить сопровождение системы, потому что как язык Ruby, так и инфраструктура Rails продолжают стремительно развиваться. А может быть, вы пишете систему веб-телефонии, и вам нужно заставить телефон «звонить». Если для выполнения требований к производительности сервер должен передавать веб-странице полноценные асинхронные события, то архитектура, основанная на сервлетах, упростит тестирование и внесение изменений.

В реальных проектах удовлетворение потребностей заинтересованных сторон требует принятия многочисленных решений, которые отнюдь не сводятся к выбору веб-технологии. Действительно ли вам нужна «архитектура» и действительно ли вам нужен «архитектор» для принятия решений? Кто должен принимать эти решения? Программист, который может принять многие решения неосознанно и косвенно, или архитектор, который принимает их абсолютно сознательно, четко представляя себе систему в целом, путь ее возможной эволюции и всех заинтересованных сторон? Как бы то ни было, у проекта появится архитектура. Будет ли она явно разрабатываться и документироваться, или же архитектура будет подразумеваться и, чтобы понять ее, придется читать программный код?

Конечно, часто выбор оказывается не столь бескомпромиссным. Но с ростом самой системы, ее сложности и количества работающих над ней людей эти ранние решения и способ их документирования начинают играть все более важную роль.

Надеемся, вы понимаете, насколько важны архитектурные решения для выполнения качественных требований, и примете эти решения сознательно, не полагаясь на то, что «архитектура сформируется сама собой по ходу дела».

А если система очень велика? Одна из причин, по которой мы применяем архитектурные принципы (скажем, принцип «разделяй и властвуй»), заключается в сокращении сложности и возможности параллельной работы. Это позволяет нам создавать все более крупные системы. Можно ли разделить саму архитектуру на части, над которыми будут параллельно работать разные люди? Джеррит Блау и Фред Брукс пишут о компьютерных архитектурах следующее:

> «...если и после применения всех приемов, направленных на то, чтобы задача стала постижимой для одного человека, архитектура все еще остается слишком масштабной и сложной, значит, планируемый продукт

> слишком сложен, и от его создания следует отказаться. Иначе говоря, компьютерная архитектура должна быть сложной настолько, чтобы ее мог понять один человек. Если один человек не способен спроектировать запланированную архитектуру, то один человек также не сможет ее понять.» (1977)

Действительно ли необходимо понимать все аспекты архитектуры для ее использования? Архитектура обеспечивает разделение ответственности, так что в большинстве случаев разработчику или тестеру, использующему архитектуру для построения или сопровождения системы, не обязательно иметь дело с архитектурой в целом – им достаточно взаимодействовать только с теми частями, которые необходимы для выполнения заданной функции. Это позволяет нам создавать системы, непостижимые для разума одного человека. Но прежде чем полностью игнорировать совет людей, построивших IBM System/360, одну из самых долговечных компьютерных архитектур, давайте разберемся, что заставило их выступить с таким утверждением.

Фред Брукс говорит, что концептуальная целостность является самым важным атрибутом архитектуры: «Лучше, если система... отражает одну совокупность идей проектирования, чем если она содержит много хороших, но независимых и нескоординированных идей»[1] (1995). Именно концептуальная целостность позволяет разработчику, уже знакомому с одной частью системы, быстро разобраться в другой части. Концептуальная целостность образуется из логической последовательности в таких областях, как критерии декомпозиции, применение шаблонов проектирования и форматы данных. Это позволяет разработчику применить опыт, полученный в ходе работы над одной частью системы, для разработки и сопровождения других частей системы. В системе действуют единые правила. При переходе от системы к «системе систем» концептуальная целостность также должна сохраняться и в архитектуре интеграции систем – например, выбором определенного архитектурного стиля (например, *канал сообщений «публикация/подписка»*) и последовательным применением этого стиля для интеграции всех систем в «систему систем».

Проблема архитектурной группы состоит в том, чтобы сохранить единую философию и единство мысли при создании архитектуры. Ограничьте группу минимальным количеством участников, организуйте высокий уровень сотрудничества с частым общением между участниками и назначьте одного-двух «благожелательных диктаторов» с правом последнего слова во всех решениях. Такая организационная схема

[1] Ф. Брукс. «Мифический человеко-месяц», Символ-Плюс, 2000.

часто встречается в успешных системах (как коммерческих, так и распространяемых с открытым кодом), а ее применение обеспечивает концептуальную целостность, которая является одним из непременных атрибутов красивых архитектур.

Хорошими архитекторами часто становятся люди, которые учатся у еще лучших архитекторов (Waldo 2006). Возможно, это объясняется тем, что некоторые концептуальные требования присущи практически всем проектам. Некоторые из них уже упоминались ранее, но ниже приводится более полный список. Каждое концептуальное требование сформулировано в виде вопроса, который архитектор должен задать себе в ходе проекта. Конечно, у некоторых систем могут существовать свои, дополнительные критические концептуальные требования.

Функциональность

Какую функциональность продукт предложит своим пользователям?

Способность к изменениям

Какие изменения могут потребоваться в программном продукте в будущем? Какие изменения маловероятны (а, следовательно, вам не нужно обеспечивать их простое внесение в будущем)?

Производительность

Какую производительность должен обеспечивать продукт?

Емкость

Сколько пользователей будет одновременно работать с системой? Какой объем данных система должна хранить для своих пользователей?

Экосистема

Как система будет взаимодействовать с другими системами в экосистеме, в которой она будет развернута?

Модульность

Как задача написания программного продукта делится на рабочие задания (модули) – и особенно модули, которые могут разрабатываться независимо друг от друга, при этом легко и точно дополняя потребности друг друга?

Удобство построения

Может ли программный продукт строиться в виде набора компонентов, реализуемых и тестируемых независимо друг от друга? Какие компоненты можно позаимствовать из других продуктов, а какие придется приобретать у внешних поставщиков?

Технологичность

Если продукт существует в нескольких версиях, то как он может разрабатываться в контексте линейки продуктов, с использованием общности версий? По какой стратегии должны разрабатываться продукты, входящие в линейку (Weiss and Lai 1999)? Какие капиталовложения потребуются на создание линейки программных продуктов? Какую предполагаемую прибыль принесет разработка разных продуктов, входящих в линейку?

В частности, возможно ли начать с разработки минимально полезного продукта, а затем разрабатывать дополнительные продукты линейки посредством добавления (и удаления) компонентов без изменения ранее написанного кода?

Безопасность

Требует ли использование продукта авторизации, должен ли продукт ограничивать доступ к своим данным, как обеспечить безопасность данных? Как защититься от атак «отказа в обслуживании» (DoS, Denial of Service) и других видов атак?

Наконец, хороший архитектор понимает, что архитектура влияет на организацию. Конуэй заметил, что структура системы отражает структуру той организации, которая ее построила (1968). Однако архитектор знает, что закон Конуэя может работать и в обратном смысле. Другими словами, хорошая архитектура может повлиять на структуру организации, чтобы повысить ее эффективность при построении систем на базе этой архитектуры.

Архитектурные структуры

Как же хороший архитектор справляется со всеми этими требованиями? Ранее мы уже упоминали о необходимости деления системы на структуры, каждая из которых определяет конкретные отношения среди определенных типов компонентов. Основная задача архитектора – структурировать систему таким образом, чтобы каждая структура помогала ответить на определяющие вопросы одного из концептуальных требований. Ключевые структурные решения разделяют продукт на компоненты и определяют отношения между этими компонентами (Bass, Clements, and Kazman 2003; Booch, Rumbaugh, and Jacobson 1999; IEEE 2000; Garlan and Perry 1995). Для любого продукта существует много структур, которые необходимо спроектировать. Каждая структура проектируется по отдельности, чтобы она могла рассматриваться как отдельное концептуальное требование. В нескольких ближайших разделах рассматриваются некоторые структуры, которые мо-

гут использоваться для требований из списка. Например, структуры сокрытия информации показывают, как система делится на рабочие задания. Они также могут использоваться в качестве плана изменений: для каждого предполагаемого изменения указывается, в какие модули эти изменения будут вноситься. Для каждой структуры описываются компоненты и отношения между ними, определяющие структуру. Для концептуальных требований из нашего списка самыми важными мы считаем именно перечисленные ниже структуры.

Структуры сокрытия информации

Компоненты и отношения: основными компонентами являются модули сокрытия информации. Каждый модуль представляет собой рабочее задание для группы разработчиков и воплощает проектировочное решение. Мы говорим, что проектировочное решение является секретом модуля, если оно может быть изменено без последствий для других модулей (Hoffman and Weiss 2000, главы 7 и 16). Основное отношение между модулями выражается формулировкой «является частью». Модуль сокрытия информации A является частью модуля сокрытия информации B, если секрет модуля A является частью секрета модуля B. Обратите внимание: секрет A должен изменяться без изменения других частей B; в противном случае A не является подмодулем согласно нашему определению. Например, во многих архитектурах имеются модули виртуальных устройств, секретом которых является взаимодействие с некоторыми физическими устройствами. Если виртуальные устройства делятся на типы, то каждый тип может образовывать подмодуль модуля виртуальных устройств. В этом случае секретом каждого типа виртуальных устройств является взаимодействие с устройствами данного типа.

Каждый модуль – это рабочее задание, включающее набор программ, которые должны быть написаны разработчиками. В зависимости от языка, платформы и среды «программой» может быть метод, процедура, функция, сценарий, макрос или другая последовательность команд, выполняемая на компьютере. Вторая структура модулей сокрытия информации основана на отношении «содержится в» между программами и модулями. Программа P содержится в модуле M, если написание P является частью рабочего задания M. Обратите внимание: каждая программа содержится в некотором модуле, потому что каждая программа должна быть частью рабочего задания некоторого разработчика.

Одни программы доступны через интерфейс модуля, другие используются в его внутренних операциях. Отношения между модулями также могут устанавливаться через интерфейсы. Интерфейс модуля состоит из совокупности допущений программ, внешних по отношению к мо-

дулю, относительно самого модуля, и совокупности допущений программ модуля относительно программ и структур данных других модулей. Говорят, что A «зависит от» интерфейса B, если изменение в интерфейсе B может потребовать изменений в A.

Структура «является частью» образует иерархию. Конечными узлами этой иерархии являются модули, не содержащие идентифицированных подмодулей. Структура «содержится в» также образует иерархию, поскольку каждая программа содержится только в одном модуле. Отношение «зависит от» не всегда образует иерархию, потому что два модуля могут зависеть друг от друга либо напрямую, либо через более длинную циклическую цепочку отношений «зависит от». Обратите внимание: отношение «зависит от» не следует путать с отношением «использует» (см. далее).

Структуры сокрытия информации образуют основу парадигмы объектно-ориентированного проектирования. Если модуль сокрытия информации реализуется в виде класса, то открытые методы класса входят в интерфейс модуля.

Концептуальные требования: структуры сокрытия информации должны проектироваться таким образом, чтобы они обеспечивали способность к изменениям, модульность и удобство построения.

Структуры «использует»

Компоненты и отношения: в соответствии с приведенным ранее определением модули сокрытия информации содержат одну или несколько программ (см. предыдущий раздел). Две программы включаются в один модуль в том и только в том случае, если они имеют общий секрет. Компонентами структуры «использует» являются программы, которые могут активизироваться независимо друг от друга. Обратите внимание: эти программы могут вызываться друг другом или оборудованием (например, функции обработки прерываний), вызов может поступить от программы из другого пространства имен (например, функции операционной системы или удаленные процедуры). Более того, вызов может происходить на любой стадии, от компиляции до времени выполнения.

Формирование структуры «использует» следует рассматривать только между программами, работающими на одной стадии привязки. Вероятно, начинать проще с программ, связываемых на стадии выполнения. Позднее также можно подумать о формировании отношений типа «использует» между программами на стадии компиляции или загрузки.

Мы говорим, что программа A использует программу B, если присутствие программы B и ее соответствие своей спецификации необходимо