Оглавление

Предисловие

Долгое время разработчики программного обеспечения придерживались мнения, что архитектуру программного обеспечения следует разрабатывать до написания первой строки кода. Под влиянием строительной отрасли считалось, что признаком успешной архитектуры программного обеспечения было то, что в ней не надо ничего менять в процессе разработки, это часто было связано с высокими затратами на переделку архитектуры.

В дальнейшем с появлением методов гибкого программного обеспечения такое представление об архитектуре изменилось. Метод заранее спланированной архитектуры основывался на том, что установленные требования не должны были меняться до написания кода, что приводило к поэтапному (или каскадному) методу проектирования, в котором за установленными требованиями следовала разработка архитектуры, а за ней, в свою очередь, следовала разработка программного обеспечения. Однако динамично меняющийся мир поставил под сомнение саму идею неизменности требований, так как изменения требований оказались просто необходимы для ведения бизнеса в современном мире и обеспечивали планирование проекта, позволяющее вносить контролируемые изменения.

В этом новом гибком мире многие люди стали задаваться вопросом о роли архитектуры. И конечно же, заранее планируемая архитектура не могла приспособиться к изменчивости современного мира. При создании архитектуры используется другой метод, который

гибким образом вносит необходимые изменения. При таком подходе разработка архитектуры выполняется в тесной связи с созданием программного обеспечения, так что архитектура может не только реагировать на изменение требований, но также принимать во внимание обратную связь от программирования. Мы обратимся к такой архитектуре с эволюционным развитием, чтобы показать, что даже в случае непредсказуемых изменений эта архитектура все еще может продолжать двигаться в правильном направлении.

Работая в компании *ThoughtWorks*, мы глубоко окунулись в это архитектурное мировоззрение. Ребекка вела многие из наших наиболее важных проектов в 2000-х и постепенно обеспечивала лидирующее положение, будучи руководителем технического отдела. Нил играл роль внимательного наблюдателя наших работ, синтезируя и передавая полученный нами опыт. Патрик совмещал свою работу над проектом с обеспечением нашего лидирующего положения в области техники. Мы всегда чувствовали жизненную важность создаваемой нами архитектуры и не могли довериться случаю. Совершая ошибки, мы получали бесценный опыт и начинали лучше понимать, как строить основу кода, которая могла бы эффективно реагировать на многие изменения своей цели.

В основе создания архитектуры с эволюционным развитием лежат небольшие изменения, которые с помощью обратной связи позволяют каждому узнать, как развивается система. Появление системы непрерывной поставки стало решающим фактором практической реализации архитектуры с эволюционным развитием. Авторское трио Нила, Ребекки и Патрика использует идею функций пригодности для управления состоянием архитектуры. Они изучают различные стили эволюционируемости архитектуры и делают акцент на проблемах, связанных с данными длительного хранения, — обычно этим пренебрегают. В большинстве пояснений, приведенных в этой книге, используется закон Конвея.

Несмотря на то что мы еще должны многое узнать о разработке архитектуры программного обеспечения эволюционирующего типа, эта

книга знаменует собой важную веху, обозначающую текущее состояние понимания указанной проблемы. По мере того как все больше людей начинают осознавать роль систем программного обеспечения в XXI веке, знание того, как лучше всего реагировать на изменения, сохраняя достигнутое, будет важнейшим навыком в области создания программного обеспечения.

— Мартин Фаулер
martinfowler.com
Сентябрь 2017 г.

Введение

Типографские соглашения

В этой книге приняты следующие типографские соглашения:

Курсив

Используется для обозначения новых терминов.

`Моноширинный шрифт`

Используется для оформления листингов программ и программных элементов внутри обычного текста, таких как имена переменных, функции, базы данных, типы данных и переменных окружения, инструкции и ключевые слова.

Так выделяются советы и предложения.

От научного редактора перевода

Эта книга как нельзя лучше подойдет новичкам в области архитектуры ПО, которые хотят повысить свою квалификацию. «Эволюционная архитектура» хорошо расширяет кругозор, так как в ней содержится большой объем информации без лишних усложнений.

Начинающего разработчика книга вдохновит на создание архитектуры, которая будет соответствовать концепциям авторов.

Практикующий опытный специалист, в своих проектах уже имевший дело со слоистой архитектурой, возможно, скептически отнесется к идеям, изложенным в этом издании, но через пару месяцев после прочтения переосмыслит свой подход и даже попробует что-то внедрить. А при переходе в новый проект наверняка попытается построить архитектуру с возможностью эволюционирования. Ведь никому не хочется в сотый раз проектировать одно и то же.

Именно такой подход описывают Нил Форд, Ребекка Парсонс и Патрик Куа, чтобы вы захотели и смогли перевести IT-отрасль на эволюционную архитектуру.

Как связаться с нами

С вопросами и предложениями, касающимися этой книги, обращайтесь в издательство:

O'Reilly Media, Inc.

1005 Gravenstein Highway North Sebastopol, CA 95472

800-998-9938 (США или Канада)

707-829-0515 (международный и местный)

707-829-0104 (факс)

Список опечаток, файлы с примерами и другую дополнительную информацию вы найдете на странице книги http://oreil.ly/2eY9gT6.

Свои пожелания и вопросы технического характера отправляйте по адресу bookquestions@oreilly.com.

Дополнительную информацию о книгах, обсуждения, конференции и новости вы найдете на веб-сайте издательства: http://www.oreilly.com.

Ищите нас в Facebook: http://facebook.com/oreilly.

Следуйте за нами в «Твиттере»: http://twitter.com/oreillymedia.

Смотрите нас на YouTube: http://www.youtube.com/oreillymedia.

Дополнительная информация

Авторы ведут сопутствующий сайт этой книги: http://evolutionary architecture.com.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Благодарности

Нил хотел бы выразить благодарность всем участникам конференций, на которых он выступал за последние несколько лет и которые помогли ему уточнить и исправить приведенный в книге материал. Он также хотел бы поблагодарить технических рецензентов, которые предоставили превосходную обратную связь и рекомендации. Особая благодарность редакторам Венкату Субраманиуму (Venkat Subramanium), Оуэну Вудсу (Eoin Woods), Саймону Брауну (Simon

Brown) и Мартину Фаулеру (Martin Fowler). Нил также хотел бы поблагодарить своих котов Уинстона, Паркера и Изабеллу за полезные перерывы, которые всегда приводят к озарениям. Он благодарит своего друга Джона Дресчера (John Drescher) и его коллег из компании *ThoughtWorks*; Нормана Запьена (Norman Zapien) за его проницательность, его группу *Pasty Geeks* и расположенный по соседству *Cocktail Club* за дружбу и поддержку. И наконец, ему хотелось бы поблагодарить свою многострадальную жену, которая с улыбкой переносит путешествия Нила и другие издержки профессии.

Ребекка хотела бы поблагодарить всех своих коллег, участников конференций и докладчиков, а также авторов, которые на протяжении многих лет генерировали идеи, предоставляли инструменты и методы, задавая уточняющие вопросы об этой области и архитектуре с эволюционным развитием. Вслед за Нилом она выражает благодарность техническим рецензентам за тщательное чтение и комментарии. Ребекке также хотелось бы поблагодарить соавторов за все проливающие свет разговоры и обсуждения в процессе совместной работы над этой книгой. В частности, она благодарна Нилу за полезные обсуждения и даже, может быть, споры, которые велись несколько лет в отношении различий между независимой архитектурой и архитектурой с эволюционным развитием. Эти идеи прошли длинный путь с момента первого разговора.

Патрику хотелось бы поблагодарить всех коллег и клиентов компании *ThoughtWorks*, которые направляли эту работу в нужное русло и предоставили испытательную среду для координации идей в построении архитектуры с эволюционным развитием. Ему бы также хотелось присоединиться к благодарностям Нила и Ребекки техническим рецензентам, которые помогли значительно улучшить эту книгу. И наконец, он хотел бы поблагодарить соавторов за возможность совместной работы над этой книгой, несмотря на большую разницу в часовых поясах и редкие личные встречи.

1 Архитектура программного обеспечения

Долгое время разработчики вели борьбу за краткое определение архитектуры программного обеспечения, так как это достаточно большая и постоянно меняющаяся область. Ральф Джонсон превосходно определил архитектуру программного обеспечения как «важную штуку (чем бы это ни было)». Работа разработчика архитектуры состоит в том, чтобы понять и сбалансировать все эти важные вещи (чем бы они ни были).

Важная часть разработки архитектуры состоит в том, чтобы понять требования той или иной сферы бизнеса или объекта к предлагаемому решению. Хотя эти требования работают как мотивация использования программного обеспечения для решения поставленной задачи, они в конечном счете представляют собой лишь один из факторов, которые разработчики архитектуры (архитекторы) должны учитывать при создании архитектуры. Разработчики архитектуры должны также учитывать многие другие факторы. Некоторые факторы явные (такие, как соглашения об уровне предоставления услуг), а другие неявные для той или иной области бизнеса (например, компания занимается слиянием и поглощением других компаний). Поэтому качество архитектуры программного обеспечения проявляется в способности ее разработчика анализировать бизнес-требования и требования предметной области наряду с другими важными факторами, чтобы найти решение, которое оптимально уравновешивало бы все проблемы. Область архитектуры программного обеспечения образуется из объединения всех этих факторов, как это показано на рис. 1.1.

Рис. 1.1. Вся область архитектуры окружена требованиями и различными возможностями

Как видно на рис. 1.1, требования предметной области существуют вместе с другой функциональностью архитектуры, определяемой разработчиком. Сюда входит широкий диапазон внешних факторов, которые могут изменять процесс решения в отношении того, как строить систему программного обеспечения. Для примера проверьте приведенный в табл. 1.1 список.

Таблица 1.1. Частичный перечень возможностей

доступность	идентифицируемость	точность	адаптивность	администрируемость
ценовая доступность	гибкость	контролируемость	автономность	пригодность
совместимость	сочетаемость	конфигурируемость	корректность	достоверность
настраиваемость	способность к разделению	определяемость	демонстрируемость	функциональная надежность
способность к отладке	способность к развертыванию	открытость	распределяемость	продолжительность
эффектность	эффективность	пригодность к использованию	способность наращивания	прозрачность отказов

отказоустойчивость	верность передачи информации	универсальность	возможность контроля	возможность установки целостности
совместимость	обучаемость	восстанавливаемость	управляемость	мобильность
модифицируемость	модульность	работоспособность	независимость	портативность
точность	предсказуемость	возможность оценки	продуктивность	доказуемость
восстанавливаемость	значимость	надежность	повторяемость	воспроизводимость
устойчивость к внешним возмущениям	быстрота реагирования	возможность повторного использования	прочность	безопасность
масштабируемость	незаметность для пользователя	автономность	удобство обслуживания	способность обеспечения безопасности
простота	стабильность	соответствие стандартам	живучесть	устойчивость
легкость сопровождения	способность получения заданных свойств	тестируемость	своевременность	отслеживаемость

При создании программного обеспечения архитектор должен определить самые важные из этих возможностей. Однако многие из этих факторов противодействуют друг другу. Например, достижение высокой производительности и высокой масштабируемости может быть затруднительно, потому что достижение того и другого требует точного баланса архитектуры, операций и многих других факторов. В результате необходим анализ проекта архитектуры, а неизбежное столкновение конкурирующих факторов требует баланса, но балансировка за и против каждого решения в проектировании архитектуры ведет к *компромиссам,* на которые так часто жалуются разработчики архитектуры. В последние несколько лет инкрементная разработка программного обеспечения создала фундамент для переосмысления того, как архитектура меняется со временем, и того, как можно за-

щитить в ходе эволюционного развития важные характеристики архитектуры. Эта книга связывает между собой новый взгляд на *архитектуру* и *время*.

Нам хотелось бы добавить новую возможность в архитектуру программного обеспечения, а именно *способность к эволюционному развитию*.

Архитектура с эволюционным развитием

Несмотря на все наши усилия, программное обеспечение со временем становится все сложнее менять. По разным причинам те части, которые составляют системы программного обеспечения, не позволяют выполнять простые изменения и становятся со временем все более хрупкими и неподатливыми. Изменения в проектах по разработке программного обеспечения обычно обусловлены повторной оценкой функциональности и/или границами области применения. Изменения другого типа происходят вне области контроля архитекторов и специалистов по долгосрочному планированию. Хотя разработчикам архитектуры нравится чувствовать себя способными к стратегическому планированию, постоянно меняющаяся среда разработки программного обеспечения делает это затруднительным. Поскольку мы не можем избежать изменений, то нам необходимо их использовать.

Как можно осуществлять долгосрочное планирование, если все постоянно меняется?

В биологическом мире окружающая среда непрерывно меняется под воздействием природных и антропогенных причин. Например, в начале 1930-х годов в Австралии были проблемы с жуками-носорогами, которые вредили побегам сахарного тростника и делали его выращивание и сбор невыгодными. В июне 1935 года в ответ на это существовавшее в то время Бюро управления экспериментальными станциями по выращиванию сахарного тростника стало использовать для борьбы с жуками тростниковую жабу, которая раньше населяла

Южную и Среднюю Америку[1]. После разведения в неволе некоторое число молодых тростниковых жаб в июле и августе 1935 года было выпущено в Северном Квинсленде. Обладающая ядовитой кожей и не имеющая врагов, тростниковая жаба широко распространилась; на сегодняшний день их число достигло 200 миллионов. Мораль: внесение изменений в высокодинамичную экосистему может привести к непредсказуемым результатам.

Экосистема разработки программного обеспечения включает в себя все рабочие инструменты, фреймворки, библиотеки и практический опыт, накопленный в процессе разработки. Эта экосистема формирует равновесие, во многом напоминающее биологическую систему, которое разработчики могут понимать и строить внутри него. Однако это равновесие динамичное, сопровождающееся непрерывным появлением новых вещей, нарушающих баланс, пока не установится новое равновесное состояние. Представим себе перевозящего коробки моноциклиста: эта система *динамичная,* так как старается сохранять вертикальное положение, и *равновесная,* потому что продолжает поддерживать баланс. В среде разработки программного обеспечения каждое нововведение или новая практика может нарушить статус-кво, вынуждая заново достигать равновесия. Мы продолжаем подбрасывать моноциклисту еще больше коробок, вынуждая его всякий раз восстанавливать равновесие.

Разработчики архитектуры многим напоминают нашего злополучного моноциклиста, непрерывно балансирующего и адаптирующегося к изменяющимся условиям. Практика проектирования непрерывной поставки представляет собой такой тектонический сдвиг в равновесном состоянии: встраивание ранее разрозненных функций (таких, например, как операции) в цикл разработки программного обеспечения открывает новые перспективы на понятие *изменений*. Разработчики архитектуры корпоративных приложений больше не могут опираться на пятилетние планы, потому что вся концепция разработки про-

[1] Clarke, G. M., Gross, S., Matthews, M., Catling, P. C., Baker, B., Hewitt, C. L., Crowther, D., & Saddler, S. R. 2000, Environmental Pest Species in Australia, Australia: State of the Environment, Second Technical Paper Series (Biodiversity), Department of the Environment and Heritage, Canberra.

граммного обеспечения видоизменится за этот период, потенциально превращая долгосрочные решения в категорию спорных.

Разрушительные изменения сложно прогнозировать даже опытным практикам. Появление инструмента композиции контейнеров Docker (https://www.docker.com/) является примером непостижимого сдвига в промышленности. Однако мы можем проследить появление контейнеризации с помощью серии небольших последовательных шагов. Некоторое время назад операционные системы, серверы приложений и прочие элементы инфраструктуры были субъектами коммерческой деятельности, требующими лицензирования и больших затрат. Многие архитектуры, проектировавшиеся в эти годы, были ориентированы на эффективное применение совместно используемых ресурсов. Постепенно операционная система Linux стала вполне приемлемой для многих предприятий, и *денежные* затраты на операционные системы сократились до нуля. Затем DevOps начали практическое применение автоматической инициализации машин с помощью таких инструментов, как Puppet (https://www.puppet.com/) или Chef (https://www.chef.io/), сделав ОС Linux *эксплуатационно* свободной. После того как экосистема стала свободной и нашла широкое применение, стала неизбежной консолидация распространенных форматов, используемых для переноса документов; появился Docker. Но контейнеризация не смогла бы появиться без всех эволюционных шагов, ведущих к этому завершению.

Программные платформы, которые мы используем, служат примером непрерывной эволюции. Новые версии языков программирования предлагают лучшие прикладные программные интерфейсы (API — application programming interfaces) для повышения универсальности или для области применения при решении новых задач; новые языки программирования предлагают разные парадигмы и разные наборы конструирования. Например, язык Java был представлен как замена языка C++, чтобы облегчить написание сетевого кода и улучшить процесс управления памятью. Если посмотреть на прошедшие 20 лет, мы заметим, что многие языки все еще непрерывно развивают API, в то время как у абсолютно новых языков программирования периодиче-

ски возникают новые проблемы. Эволюция языков программирования показана на рис. 1.2.

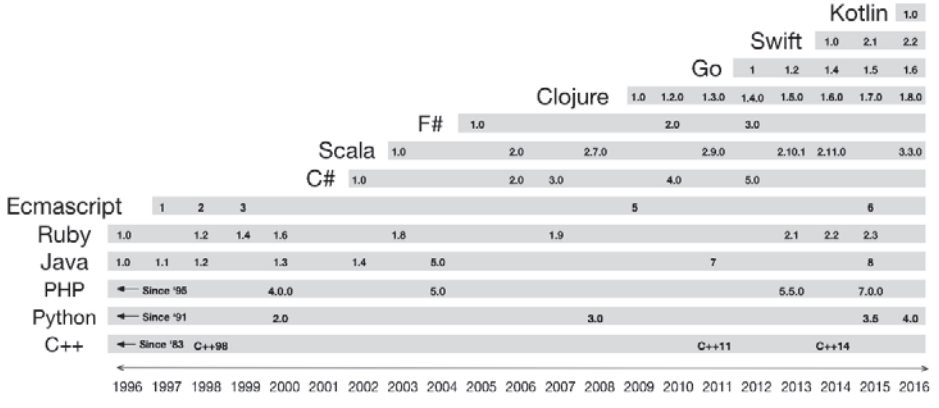

Рис. 1.2. Эволюция популярных языков программирования

Независимо от того, какой именно аспект разработки программного обеспечения мы рассматриваем — программные платформы, языки, операционную среду, технологию поддержки постоянного хранения данных и т. п., — мы ожидаем постоянных изменений. Хотя мы не можем предсказать, когда произойдут изменения в технической области или области применения или какие именно изменения сохранятся, мы знаем, что изменения неизбежны. Следовательно, мы должны разрабатывать архитектуру нашей системы, зная, что изменится техническая область.

Если экосистема непрерывно меняется непредсказуемым образом и прогнозирование изменений невозможно, то что тогда является *альтернативой* жесткому планированию? Архитекторы корпоративных приложений и другие разработчики должны научиться адаптироваться. Частью традиционного обоснования, лежащего в основе долгосрочного планирования, были вопросы финансирования; изменения программного обеспечения были дорогостоящими. Однако современная практика проектирования свела на нет этот фактор, сделав изменения менее затратными путем автоматизации

выполнявшихся вручную в прошлом процессов и в результате появления других практик, таких как DevOps (development и operations).

Многие разработчики микропроцессорных систем столкнулись с тем, что некоторые части их систем было сложнее модифицировать, чем остальные. Вот почему *архитектуру программного обеспечения* определяют как «часть системы, которую сложно будет менять потом». Это удобное определение все делило на те части, которые *можно* изменить без особых усилий, и на те, которые менять действительно сложно. К сожалению, это определение превращается в «белое пятно», когда начинаешь думать об архитектуре: предположение разработчиков, что изменения сложно внести, становится самосбывающимся пророчеством.

Несколько лет назад некоторые разработчики инновационных архитектур программного обеспечения переосмыслили проблему «сложно будет менять позже» в новом свете: что, если возможность менять мы встроим *в* архитектуру? Другими словами, если *простота изменения* является базовым принципом архитектуры, тогда вносить изменения будет несложно. Встраивание в архитектуру способности эволюционировать в свою очередь приводит к появлению целого ряда новых характеристик, снова нарушая динамические характеристики.

Даже если экосистема не меняется, то что можно сказать насчет постепенного ухудшения характеристик архитектуры? Разработчики проектируют архитектуры, но затем подвергают их воздействию хаотичного реального мира, *внедряющего* нечто поверх архитектуры. Каким образом разработчики архитектуры защищают важные части, которые они таковыми определили?

Как можно защитить созданную архитектуру от постепенной деградации?

Постепенное ухудшение, которое часто называют *битовой деградацией* (bit rot), возникает во многих организациях. Разработчики архитектуры выбирают определенный паттерн архитектуры для