Публикация пакетов Python

Тестирование, распространение и автоматизация проектов

Дэйн Хиллард

БОМБОРА
ИЗДАТЕЛЬСТВО

MANNING

УДК 004.43
ББК 32.973.26-018.2
Х45

Dane Hillard
PUBLISHING PYTHON PACKAGES

Хиллард, Дэйн.

Х45 Публикация пакетов Python. Тестирование, распространение и автоматизация проектов / Дэйн Хиллард ; [перевод с английского В. Краснянской]. — Москва : Эксмо, 2024. — 288 с. — (Мировой компьютерный бестселлер).

ISBN 978-5-04-189146-6

Книга «Публикация пакетов Python» описывает практический процесс масштабируемого совместного использования кода Python с высокой эффективностью и помогает получить опыт работы с новейшими инструментами упаковки. Пособие дает возможность изучить все тонкости тестирования и непрерывной интеграции пакетов, а также предлагает профессиональные советы по созданию поддерживаемого проекта с открытым исходным кодом, включая вопросы лицензирования, документации и создания сообщества участников.

УДК 004.43
ББК 32.973.26-018.2

ISBN 978-5-04-189146-6

Содержание

ЧАСТЬ 2. СОЗДАНИЕ ЭФФЕКТИВНОГО ПАКЕТА

ЧАСТЬ 3. ВЫХОД НА ПУБЛИКУ

ЧАСТЬ 4. ДОЛГИЙ РЕЙС

О книге

«Публикация пакетов Python» рассказывает о нескольких направлениях, связанных конкретно с оформлением пакетов в Python, а также о нескольких концепциях, приложимых практически к любому языку программирования. Вместе они позволяют командам и отдельным разработчикам более эффективно осуществлять поставку программного обеспечения. Команды разработки, рабочие группы и инженеры SRE найдут здесь новые полезные в работе методы и инструменты. Если вы хотите автоматизировать, стандартизировать и организовать как можно больше своих проектов Python, возможно, эта книга для вас.

Кому стоит прочитать книгу?

«Публикация пакетов Python» предназначена для любого читателя, уже знакомого с языком программирования Python и имеющего желание поделиться своим кодом с друзьями, коллегами или незнакомыми людьми во всем мире. Приемы, описанные в этой книге, отобраны так, чтобы их мог использовать один человек, но приложимы и к работе команды практически любого размера. Сотрудничество — ключ к эффективной разработке программного обеспечения, так что эти методы способствуют избавлению от утомительного

однообразия и появлению возможности сосредоточиться на таком необходимом общении с помощью кода и текста.

По мере того как в научном сообществе растет роль программного обеспечения, повышается ценность пакетов программ. Успешные проекты с открытым исходным кодом использовались в последних знаковых достижениях науки, таких как посадка на Марс или фотографии черных дыр. Ищете ли вы крупный проект или просто хотите убедиться, что PI в вашей лаборатории может проверять код, который вы используете для получения результатов, воспроизводимые процессы — именно то, что может вам помочь.

Если ранее вы не работали с такими инструментами оценки качества программного обеспечения, как модульное или юнит-тестирование и линтеры, эта книга поможет подробнее ознакомиться с автоматизацией работы и улучшить проверку качества кода в высокоавтоматизированных модулях. Вам выпал шанс подумать о выявлении проблем до того, как они разрастутся, вместо того чтобы постоянно заниматься ликвидацией разгорающихся пожаров.

Как организована книга: схема

«Публикация пакетов Python» состоит из 11 глав, разделенных на четыре части. В части 1 рассказывается о самостоятельной ценности организованного в пакеты программного обеспечения любого рода. Часть 2 проведет вас через создание работающего пакета со всеми дополнительными компонентами, которые могут для этого понадобиться. Часть 3 погрузит в автоматизацию и сопровождение, необходимые для проектов, требующих высокой степени сотрудничества. Часть 4 покажет, как повторить процесс и привести к масштабу ваших пользователей и сотрудников.

Часть 1 «Основы» готовит почву для оформления в пакеты программ на Python и дает правильный настрой на самостоятельное создание пакета. В ней рассматриваются следующие вопросы.

- В главе 1 рассказывается, как появилось пакетирование и почему оно сохраняет свою ценность для совместного использования ПО и по сей день. Эта глава расширит понимание того, что можно отнести к пакетам, и поможет осознать, что такое оформление программ нацелено на многочисленную аудиторию.
- Глава 2 начинает рассказ об инструментах, которые можно использовать для оформления продукта в пакет.
- Глава 3 показывает подоплеку того, что означает оформление продукта Python в пакет, в том числе связанные с этим процессом файлы и метаданные, а также то, как они работают в нем.

Часть 2 «Создание эффективного пакета» превращает минимальный пакет Python в сущность с реальным поведением, которую вы сможете расширить после того, как дочитаете книгу.

- В главе 4 показано, как вводить в пакет сторонние зависимости, интерфейсы командной строки и расширения из других языков программирования.
- Глава 5 представляет инструменты модульного тестирования, осуществляющие покомпонентное тестирование, чтобы убедиться в качестве поведения вашего программного обеспечения.
- Глава 6 подробнее рассказывает о качестве, в том числе о проверке на самые распространенные ошибки, безопасность типов и единообразное форматирование кода.

Часть 3 «Выход на публику» предлагает методы, которые можно использовать где угодно, но особенно в области сотрудничества с другими разработчиками.

- Глава 7 показывает силу автоматизации и принципов постоянной интеграции, наводя на мысли, как создать петлю обратной связи с другими участниками проекта.
- Глава 8 обосновывает важность документации и показывает, как интегрировать встроенную систему автоматической документации, охватывающую и код, и текст.
- Глава 9 предлагает методы для поддержания пакета Python в актуальном состоянии на регулярной основе с минимальными усилиями так, чтобы не накапливались технические долги.

В части 4 «Долгий рейс» содержатся ответы на вопросы, куда двигаться дальше, чтобы приобрести новый набор навыков.

- Глава 10 поможет превратить описанные в предыдущих главах методы в воспроизводимый шаблон проекта разработки приложения, пригодный для использования в будущем.
- Глава 11 предлагает способы создания сообщества пользователей, разработчиков и специалистов по сопровождению для ваших проектов, бурно развивающихся с помощью процессов из предыдущих глав.

Я рекомендую читать «Публикацию пакетов Python» от корки до корки. Каждая следующая глава основывается на предыдущей, создавая по пути увлекательный ряд ключевых точек.

В дополнение к основному тексту приложения помогут вам установить инструментарий, делающий куда более приятной работу по пакетированию, как я убедился на собственном опыте.

- Приложение A поможет установить инструменты, которые упростят установку многочисленных версий Python и других языков

программирования и вызовут различные интерпретаторы Python и созданные вами виртуальные среды.

- Приложение B поможет установить независимые от проекта инструменты, необходимые для описанного в книге проекта, но также пригодные к применению с целью повышения производительности в любом проекте на Python.

О коде

В книге содержится множество примеров программного кода как в пронумерованных листингах, так и в обычном тексте. В обоих случаях код выделяется моноширинным шрифтом, чтобы отделить его от остального текста. Иногда он даже выделяется **жирным шрифтом**, чтобы подчеркнуть код, меняющийся по сравнению с предыдущими этапами, например в случаях, когда к существующей строке кода добавляется новая характеристика.

Во многих случаях оригинальный программный код отформатирован: мы добавили разрывы строк и переработали добавление отступов, чтобы приспособиться к пространству, доступному на странице книги. В редких случаях даже этого оказалось недостаточно, и в листинг пришлось включить специальные знаки для обозначения продолжения строки (➥). Дополнительно комментарии к коду часто отделены от листинга, когда код включен в текст. Многие листинги сопровождают аннотации к коду, где подчеркиваются важные понятия.

Исполняемые фрагменты кода можно скопировать из liveBook (электронной онлайн-версии книги), доступной на https://livebook.manning.com/book/publishing-python-packages. Полный код приведенных в книге примеров легко загрузить с веб-сайта Manning (www.manning.com) и с GitHub (http://mng.bz/69A5).

В каждой главе код отражает полное состояние пакета на конец главы. Я долго шел к такому выбору. Поскольку конфигурация оформления в пакеты требует точного синтаксиса и значений в ряде файлов, работать с ней сложнее, чем с обычным программированием. Чтобы не допустить разочарования и путаницы, я считаю, что лучше давать ссылки, чем организовывать код с помощью листинга.

Поскольку методы оформления в пакеты могут меняться, со временем я выкладываю обновленные версии этого кода. Я делаю это отдельно от кода, который использован в этом издании, чтобы свести к минимуму путаницу, поэтому даю ссылки на эти обновления по мере их появления.

Обсуждения на форуме liveBook

При покупке этой книги вы получаете бесплатный доступ к liveBook, онлайн-платформе Manning для чтения. Использование эксклюзивных возможностей liveBook позволяет оставлять комментарии как ко всей книге, так и к ее отдельным разделам или параграфам. На этом ресурсе очень просто делать заметки для себя, задавать технические вопросы, отвечать на них и получать помощь от автора и других пользователей. Чтобы зайти на форум, перейдите на https://livebook.manning.com/book/publishing-python-packages/discussion. Больше узнать о форумах Manning и правилах поведения на них можно на https://livebook.manning.com/discussion.

Manning обязуется обеспечить своим пользователям пространство для плодотворного диалога между отдельными читателями и между читателями и автором. При этом к автору не предъявляется требование осуществлять какой-то определенный объем активной работы, его участие остается добровольным и неоплачиваемым. Мы предлагаем вам задавать автору сложные вопросы, чтобы поддержать его интерес. Форум и архив предыдущих обсуждений остаются доступными на веб-сайте издателя, пока книга находится в печати.

Часть 1

Основы

Оформление программного обеспечения в пакеты — это, возможно, самое большое достижение, способствующее выходу приложений и логики работы на потребительский рынок. Пакеты позволяют использовать в своих проектах чужие наработки, устанавливать на телефоны приложения и многое другое. Без пакетирования пониженная работоспособность по-прежнему держала бы нас в темных веках разработки программного обеспечения.

Поддерживаете ли вы уже пакет Python или только начинаете работать с пакетированием, глубокое понимание понятий оформления ПО в пакеты настроит вас на правильный лад для работы с этой книгой и другими проектами. В этой части мы расскажем, что такое пакетирование, что необходимо для создания собственного пакета Python и что входит в минимальный рабочий пакет.

Глава 1

Зачем и почему нужны пакеты Python

Вопросы, освещенные в этой главе:

- упаковка кода делает его более доступным для других людей;
- использование пакетов делает ваши проекты значительно более управляемыми;
- создание пакетов Python для различных платформ.

Представьте, что вы написали на Python принципиально новый фрагмент программного обеспечения для беспилотных автомобилей. Ваша последняя работа изменит мир, и вы хотите, чтобы ее использовали как можно больше людей. Вы убедили компанию CarCorp* применить ваше решение, и теперь они ждут от вас код, чтобы начать с ним работать.

Представитель CarCorp звонит вам, чтобы узнать, как установить и использовать вашу программу, и вы в подробностях описываете все детали, связанные с копированием каждого файла в правильный каталог, не забывая

* Гипотетическая компания, придуманная автором. — *Прим. пер.*

упомянуть, что некоторые файлы надо сделать исполняемыми, чтобы запускать их как команды, и так далее. Поскольку вы пишете программное обеспечение, все это стало вашей второй натурой. Но, к вашему удивлению, разработчики на другом конце телефонного провода чувствуют себя растерянными. Что же произошло?

Вы обнаруживаете разрыв, часто возникающий между создателями программного обеспечения и его пользователями. Сегодня, когда им нужно что-нибудь новое, люди обычно заходят в AppStore на своем телефоне. Вам нужно проделать кое-какую работу, чтобы улучшить алгоритм взаимодействия вашего ПО с пользователем!

Из этой книги вы узнаете, как превратить свой проект на Python в устанавливаемый пакет, что делает плоды вашего труда более доступными для других людей. Также вы научитесь создавать воспроизводимый процесс, управляющий вашими проектами и уменьшающий усилия, требующиеся для их поддержки, так что сможете сосредоточиться на действительно вдохновляющей задаче — на изменении мира. Вы сделаете все это, создав реальный проект, использовав нескольких популярных инструментов для упаковки и автоматизировав некоторые составляющие процесса. Хотя сообщество Python разработало стандарты для некоторых областей пакетирования, Единственный истинный путь так и не появился. Возможно, этого не случится никогда.

Даже если вам уже приходилось создавать и публиковать пакеты Python, в этой книге вы найдете что-то новое. Предлагаемые здесь идеи и инструменты — это проверенные временем подходы к некоторым более расплывчато сформулированным методам оформления программ в пакеты. Пакетирование на Python — запутанная история со множеством альтернативных вариантов, так что вы не только увидите доступные на сегодняшний день инструменты и научитесь ими пользоваться. Вы также узнаете, какая методология легла в основу их работы, и сможете изменять их так, чтобы довести до совершенства. Исходя из этого, прежде всего нужно понять, зачем вообще пакетируют программы.

1.1. Так что же такое пакет?

Чтобы сохранить хорошие отношения с CarCorp, вы обещаете вернуться через несколько недель и представить усовершенствованный процесс, который поможет им легко установить ваше программное обеспечение. Вы знаете, что некоторые из ваших любимых библиотек Python, например Pandas или Requests, доступны как пакеты онлайн, и вы хотите обеспечить своим потребителям такую же простую инсталляцию.

Сборка пакетов — это архивирование программного обеспечения вместе с метаданными, описывающими файлы. Разработчики обычно создают такие

архивы или пакеты, намереваясь поделиться своими программами или опубликовать их.

> **ВАЖНО.** В экосистеме Python слово «пакет» используется для двух отдельных понятий. Python Packaging Authority (PyPA) различает термины в Python Packaging User Guide (https://packaging.python.org) следующим образом.
>
> - Импортируемые пакеты (Import packages) организуют многочисленные модули Python в каталог в целях разработки (http://mng.bz/wypg).
> - Распространяемые пакеты (Distribution packages), или дистрибутивы, представляют собой архивы проектов Python, предназначенные для публикации и инсталляции другими людьми (http://mng.bz/qoNz).
>
> Импортируемые пакеты не всегда распространяются в архиве, а дистрибутивы часто содержат один или несколько импортируемых пакетов. Распространяемые пакеты — основной объект изучения этой книги, и я буду отделять их от импортируемых пакетов там, где это необходимо.

Возможно, существует бесконечное количество способов соединить программное обеспечение и его метаданные. Так как же оправдать ожидания специалистов по поддержке программ и пользователей и снизить количество работы вручную? Здесь на помощь приходит система управления пакетами.

1.1.1. Стандартизация пакетов для автоматизации

Система управления пакетами, или *диспетчер пакетов,* стандартизирует архив и формат метаданных для упаковки программного обеспечения в определенной сфере деятельности. Диспетчеры пакетов предоставляют инструменты, помогающие потребителям установить зависимости на уровне проекта, языка программирования, фреймворка и операционной системы. Большинство из них поставляются с давно знакомым набором инструкций по установке, удалению или обновлению пакетов. Возможно, вам доводилось пользоваться одним из этих диспетчеров пакетов:

а) pip (https://pip.pypa.io);
б) conda (https://docs.conda.io);
в) Homebrew (https://brew.sh/);
г) npm (https://www.npmjs.com/);
д) asdf (https://asdf-vm.com/).

НА ЗАРЕ УПРАВЛЕНИЯ ПАКЕТАМИ

Хотя разработчики и раньше в течение какого-то времени неофициально оформляли код в пакеты, системы управления пакетами стали широко доступными только в начале 1990-х годов, когда этот подход обрел популярность (см. Jeremy Katz, A Brief History of Package Management, *Tidelift*, http://mng.bz/7ZG4).

Способность декларативно объявлять зависимости проекта стала преимуществом, которое повысило эффективность разработчиков, упразднив большую часть суеты в процессе управления программными проектами.

Репозитории программного обеспечения позволяют дальнейшую стандартизацию сборки пакетов, действуя как централизованные рыночные платформы для публикации и хранения пакетов, которые могут установить другие люди (рис. 1.1). Многие сообщества языков программирования обеспечивают официальные или существующие с согласия пользователей стандартные репозитории для устанавливаемых пакетов. Некоторые из популярных репозиториев программного обеспечения — это PyPI (https://pypi.org), RubyGems (https://rubygems.org/) и Docker Hub (https://hub.docker.com/).

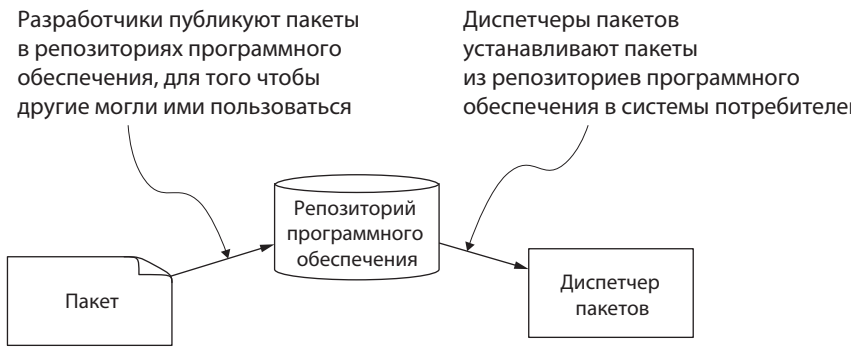

Рис. 1.1. Пакеты, диспетчеры пакетов и репозитории программного обеспечения критически важны для того, чтобы делиться программным обеспечением

Если у вас есть смартфон, планшет или обычный компьютер и вы устанавливаете на него приложения из магазина приложений, на вас работает сборка пакетов. Пакеты — это программное обеспечение, собранное вместе с метаданными о программах, и именно это и представляют собой приложения. В репозиториях программного обеспечения содержатся программы,

которые могут установить люди, и магазин приложений тоже является таким репозиторием.

Таким образом, пакеты — это программное обеспечение и метаданные, соединенные друг с другом по заранее согласованному формату и закодированные с помощью соответствующей системы управления пакетами. На более детальном уровне пакеты также обычно содержат и способ встроить программу в систему пользователя или имеют несколько предварительно подготовленных версий ПО для различных целевых систем.

1.1.2. Содержание распространяемого пакета

На рис. 1.2 показаны несколько файлов, которые можно поместить в дистрибутив. Разработчики часто включают в них код программы, но могут предоставить и скомпилированные артефакты, тестовые данные и все, что может понадобиться потребителю или коллегам. При распространении пакета потребителю предоставляется нечто вроде универсального магазина, где можно сразу приобрести все необходимое для работы со своим программным обеспечением.

Рис. 1.2. В пакет часто включают программный код, сборочный файл для компиляции кода, метаданные о коде и инструкции для потребителя

Существенные возможности предоставляет распространение не относящихся к коду файлов. Хотя последнее чаще всего и является причиной, по которой, чтобы поделиться пакетом, многим пользователям и инструментам нужны метаданные, позволяющие отличить один код от другого. В метаданных разработчики обычно указывают название своего программного проекта, его создателя (создателей), лицензию, под которой ПО можно использовать, и другие подобные данные. Обращаем особое внимание, что метаданные часто включают версию архива, чтобы разграничить ее с предыдущими и последующими публикациями проекта.

НА ЗАРЕ УПРАВЛЕНИЯ ПАКЕТАМИ

Более десяти лет после того, как стала доступна операционная система Unix, совместное пользование ПО внутри команд и среди отдельных разработчиков осуществлялось в основном вручную. Загрузка программного кода, компиляция и борьба с артефактами компиляции ложились на плечи того, кто пытался использовать код. Каждый этап этого процесса изобиловал возможностями провала из-за человеческого фактора или архитектурных или индивидуальных отличий систем. Такие инструменты, как Make (https://www.gnu.org/software/make/), убрали из процесса сборки часть подобных расхождений, но на этом и остановились, не затронув версии пакета, зависимости и управление установкой.

Теперь, когда вы познакомились с тем, что входит в пакет, обсудим, как подход к совместному пользованию программным обеспечением решает конкретные проблемы на практике.

1.1.3. Трудности совместного пользования программным обеспечением

Телефонный разговор с CarCorp напрягает вас все сильнее, и наконец вы понимаете, что забыли передать им для установки все зависимости в вашем проекте. Вы возвращаетесь на несколько шагов назад и просматриваете инсталляцию зависимостей. К несчастью, вы не помните, какую версию использовали для одной из основных зависимостей, а последняя версия, кажется, не работает. Вы проходите через установку каждой из предыдущей версий, пока наконец не находите рабочую. Кризис предотвращен. На какое-то время.

По мере усложнения разрабатываемых вами систем количество попыток, необходимых, чтобы убедиться в корректно установленной вами требуемой версии каждой зависимости, быстро увеличивается. В худших случаях можно дойти до точки, когда вам понадобятся две разные версии одной и той же зависимости, к тому же неспособные сосуществовать. Это явление с чувством называют «адом зависимостей». Вывести проект из такого состояния бывает достаточно сложно.

Но даже если ад зависимостей удается миновать, без стандартизированного подхода к сборке пакетов трудно делиться программным обеспечением обычным способом, чтобы каждый в любом месте знал, какая зависимость ему понадобится для установки вашего проекта. Сообщества программистов создают конвенции и стандарты управления пакетами, оформляя эти методы в диспетчеры пакетов, которые вы используете, чтобы делать свою работу.

Теперь, когда вы понимаете, что сборка пакетов — это то, что нужно для совместного пользования программным обеспечением, прочитайте о преимуществах, которые она дает, даже если вы не всегда хотите публиковать свои программы.

1.2. Как помогает сборка пакетов

Если вы новичок в пакетировании программ, вам может показаться, что оно существует в основном для того, чтобы находящиеся на разных концах земного шара люди могли совместно пользоваться программным обеспечением. Хотя это уже вполне веская причина для оформления кода в пакеты, вам также могут понравиться и другие преимущества, которые сборка пакетов дает при разработке программ:

а) увеличение степени сцепления (cohesion) и инкапсуляции (encapsulation);
б) более четкое определение собственности кода (ownership);
в) слабая связанность участков кода (loose coupling);
г) больше возможностей для сочетания ПО (composition).

В следующих разделах мы подробно разберем каждое из этих преимуществ.

1.2.1. Усиление сцепления и инкапсуляции с помощью пакетов

В целом отдельная часть кода должна выполнять определенную работу. *Сцепление* измеряет, насколько код соединен с этой функцией. Чем больше вокруг «шальной» функциональности, тем менее сцеплен код.

Скорее всего, вы организовываете свой код на Python, используя функции, классы, модули и импортируемые пакеты (см. Dane Hillard, The Hierarchy of Separation in Python, Chap. 2 in *Practices of the Python Pro* (Manning Publications, 2020, pp. 25–39, http://mng.bz/m2N0)). Это создает на каждой территории нечто вроде объявленных границ вокруг выполняющих определенную работу участков кода. Когда все сделано правильно, названия сообщают разработчикам, что именно находится в рамках этих границ и, что даже более важно, чего в них нет.

Несмотря на все усилия, названия и люди редко бывают идеальными. Если вы помещаете весь код на Python в одно приложение, существуют все шансы, что какая-то часть этого кода рано или поздно просочится на территории, где его не должно быть. Вспомните какой-нибудь крупный проект, который разрабатывали. Сколько раз вы создавали модули `utils.py` или `helpers.py`, содержащие целое море функциональности? Границы, которые вы устанавливали с помощью функции или модуля, легко нарушались. Эти «обслуживающие» части кода обычно привлекают другие «сервисы», из-за чего сцепление со временем уменьшается.

Представьте, что ваша система управления беспилотным автомобилем может использовать лидар (https://oceanservice.noaa.gov/facts/lidar.html) как один из способов получения входных данных. Но у транспортных средств CarCorp нет лидарных датчиков. Будучи старательным и аккуратным разработчиком, вы выделяете относящуюся к лидару часть кода, чтобы отделить ее от остальной программы. Хотя анализ наименований и правильная переработка структуры кода повысят уровень сцепления, при этом увеличится и сложность его обслуживания. Распространение пакетов ставит новые преграды на пути добавления кода туда, где его не должно быть. Поскольку обновление пакета неизбежно влечет за собой цикл упаковки, публикации и установки обновлений, оно заставляет разработчиков задумываться об изменениях, которые они делают. Вы едва ли станете добавлять код в пакет без явного намерения, оправдывающего трату времени и сил на цикл обновления.

Создание сцепления и упаковка сцепленной части кода — это путь к *инкапсуляции*. Инкапсуляция определяет, как проявляется поведение кода (и проявляется ли оно вообще), и тем самым помогает создавать у потребителей правильные ожидания от взаимодействия с вашим кодом. Вспомните проект, который вы создали и которым поделились с кем-то еще. Подумайте, сколько раз вы изменяли свой код и сколько раз менял этот код ваш партнер. Как часто этот процесс выводил его из равновесия? А вас? Инкапсуляция уменьшает подобную неразбериху, четче определяя контракт API, менее подверженный изменениям. На рис. 1.3 показано, как создавать разнообразные пакеты из связных частей кода.

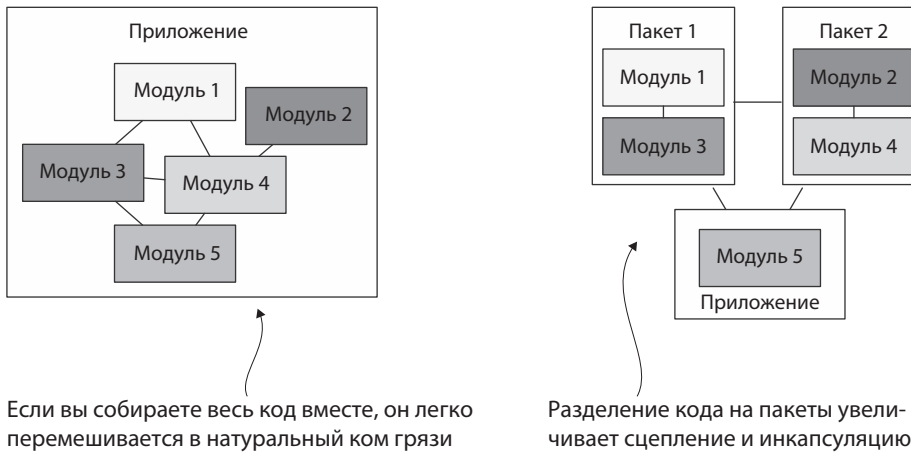

Рис. 1.3. Сборка пакетов может уменьшить непредвиденное взаимодействие между участками кода, обозначив более строгие границы

В прошлом вы наверняка раздражались, когда находили предназначенный исключительно для внутреннего использования отрывок кода в модуле, широко применяемом в программе. Такая запутанная среда ведет к ошибкам в случаях, когда не производится изменение во всех местах; таким образом вы и ваша команда теряете эффективность.

Код с хорошей инкапсуляцией и высокой степенью сцепления редко изменяется, хотя и широко используется. Его иногда называют «зрелым». Такой код — отличный кандидат для распространения пакетом, потому что вам не придется часто делать повторные публикации. Вы можете начать сборку пакета, выбрав из базы исходного кода более зрелый код, а потом применить свои знания о сцеплении и инкапсуляции, чтобы довести менее зрелый код до совершенства.

1.2.2. Четкое определение собственности кода

Командам выгодна четко установленная принадлежность участков кода конкретному разработчику. Ответственность за код часто выходит за рамки поддержания его поведения. Команды создают автоматизацию для упрощения модульного тестирования, внедрения, комплексного тестирования, тестирования рабочих характеристик и так далее. Получается слишком много стульев, на которых нужно усидеть одновременно. При небольшом объеме ограниченных участков кода команда сможет получить все эти составляющие и не сомневаться в стойкости и долговечности кода. Сборка пакетов — один из инструментов управления границами. Инкапсуляция, которую вы создаете с помощью пакетов кода, позволяет развить автоматизацию независимо от остального кода. Например, автоматизация слабо структурированной базы

исходного кода может потребовать прописывания условной логики, чтобы определить, какие тесты запускать, исходя из того, какие файлы подверглись изменению. Иначе придется после каждого изменения запускать все тесты, а это может очень замедлить работу. Создание пакетов, которые можно проверить и опубликовать независимо от остального кода, приведет к более четкому преобразованию исходного кода через тесты в код для публикации (рис. 1.4).

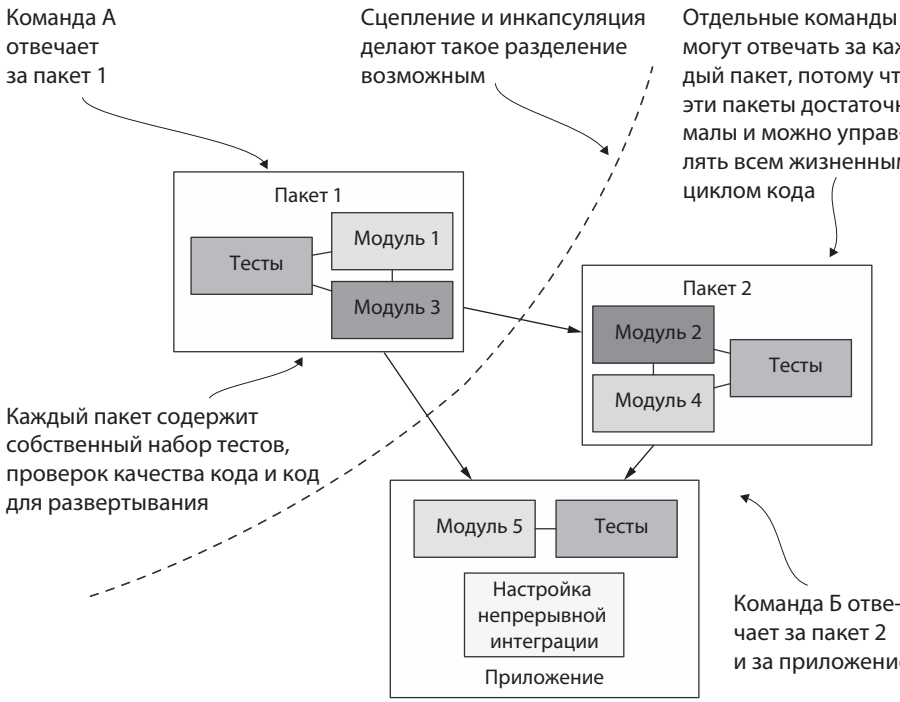

Рис. 1.4. Команды могут принимать полную ответственность за отдельные пакеты, определяя, как они хотят управлять жизненным циклом разработки, тестирования и публикации программ

Четкое ограничение цели пакета делает более вероятным очевидное разграничение ответственности. Если команда не уверена, что именно она должна делать, приняв во владение какой-то код, она будет действовать с чрезмерной осторожностью. Попытайтесь обеспечить пакет четкой областью применения, историей и руководством по эксплуатации, и вы увидите, как изменится настроение команды.

1.2.3. Отделение реализации от использования

Возможно, вы слышали термин «слабая связанность», использующийся для описания взаимозависимостей между участками кода.

> **ОПРЕДЕЛЕНИЕ.** Связанность — это мера взаимозависимости между участками кода. Более слабо связанный код обеспечивает бóльшую гибкость, что позволяет делать выбор среди многообразия стратегий исполнения и реализовывать его, вместо того чтобы следовать по одному конкретному пути. Два участка кода с низкой связанностью имеют малую взаимозависимость (или не имеют вообще никакой), так что их можно изменять в разной степени.

Сцепление и инкапсуляция, с которыми мы познакомились ранее в этой главе, — это способы снизить вероятность появления жесткой связанности из-за плохой организации кода. Код с высокой степенью сцепления имеет жесткую связанность внутри себя и слабую связанность с участками, находящимися вне его границ. Инкапсуляция позволяет открыть в API именно те функции, которые мы хотим видеть открытыми, ограничивая любую связанность с этим API. Таким образом, прибегая к сборке пакетов и инкапсуляции, вы избавляете потребителей от подробностей реализации в вашем коде. Упаковка также дает возможность освободить их от реализации в виде различных версий программы, организации пространства имен и даже языка программирования, на котором написан код.

В большом комке грязи вы все время застреваете, запуская любой код, который есть в каждом модуле. Если вы или другой член вашей команды обновляете модуль, весь использующий этот модуль код нуждается в приспособлении к изменению. Если изменение коснулось сигнатуры вызова или возвращенного значения, оно может иметь огромный радиус действия. Сборка в пакеты значительно уменьшает эту опасность (рис. 1.5).

Представьте, что каждое обновление пакета `requests` требует, чтобы вы немедленно обновили свой собственный код. Это самый настоящий ночной кошмар! Поскольку в пакетах код разделен на версии и потребители могут указать, какую именно версию они хотят установить, пакет можно обновлять неоднократно, причем эти обновления потребительского кода не касаются. Разработчики могут выбирать, когда именно приложить усилия по обновлению кода, чтобы приспособить его к изменениям в более свежей версии пакета.

Еще один способ ослабить связанность кода — это организация *пространства имен*. Данным и действиям дают названия, о чем-то говорящие пользователю. Устанавливая пакет, вы делаете его доступным в указанном им пространстве имен. Например, пакет `requests` доступен в пространстве имен `requests`.

Разные пакеты могут иметь одно и то же пространство имен. Это означает, что, если вы установите несколько таких пакетов, они могут конфликтовать, но в то же время появляется интересная возможность: гибкость

в пространстве имен означает, что пакеты могут действовать как полная альтернатива друг другу.

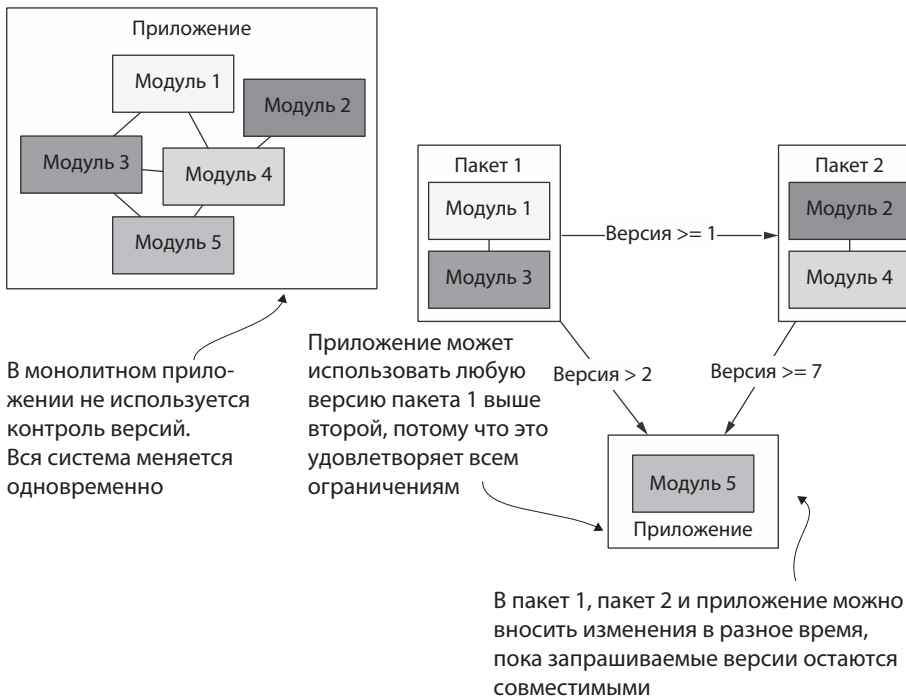

Рис. 1.5. Сборка в пакеты повышает гибкость, так что два участка кода могут развиваться с разной скоростью

Если разработчик создает новый вариант популярного пакета, более быстрый, безопасный или простой в поддержке по сравнению со старым, его можно установить на место оригинала, если только API остается прежним. Например, следующие пакеты обеспечивают примерно одинаковую клиентскую функциональность с MySQL (https://www.mysql.com). Если говорить конкретно, они обеспечивают определенный уровень совместимости с PEP 249, https://www.python.org/dev/peps/pep-0249/:

а) mysqlclient (https://github.com/PyMySQL/mysqlclient);
б) PyMySQL (https://github.com/PyMySQL/PyMySQL);
в) mysql-python (https://github.com/arnaudsj/mysql-python);
г) oursql (https://github.com/python-oursql/oursql).

Наконец, сборка пакетов Python даже способна отделить использование Python от языка, на котором написан пакет! Многие пакеты Python написаны на C и даже на Fortran, чтобы повысить производительность или облегчить интеграцию с системами предыдущего поколения. Авторы пакетов предоставляют заранее скомпилированные версии этих пакетов вместе с версиями,

которые в случае необходимости могут строиться потребителем на основе исходника. Это также делает пакеты более переносимыми, в какой-то мере освобождая разработчиков от связанных с используемым компьютером или сервером деталей. О целевых показателях сборки пакетов подробнее рассказывается в главе 3.

Возможно, вы захотите упаковать свой код, чтобы поэкспериментировать со свободным расцеплением версий и посмотреть, как ваши пакеты с версиями развиваются со временем. Быстрые изменения могут указывать на низкую степень сцепления, поскольку существует множество причин, по которым код может меняться. С другой стороны, это может указывать только на то, что код еще вызревает. По крайней мере, эти точки данных легко отслеживать! Более подробно о создании версий вы прочитаете в главе 9.

1.2.4. Заполнение позиций путем создания маленьких пакетов

Акт извлечения кода и создания множества пакетов напоминает *декомпозицию*. Успешная декомпозиция требует четкого представления о слабой связанности. Такой код является искусством, отделяющим отдельные участки кода так, чтобы их можно было соединить другим способом (удивительно краткое изложение информации о декомпозиции и связанности см. в Josh Justice, Breaking Up Is Hard to Do: How to Decompose Your Code, *Big Nerd Ranch*, http://mng.bz/5mpq).

Соединяя в пакеты малые участки кода, вы начнете выделять код, достигающий определенных целей, подлежащих обобщению или расширению для выполнения роли. Например, можно создавать одноразовые HTTP-запросы, использующие встроенную утилиту на Python, такую как `urllib.request.urlopen`. Сделав это несколько раз, вы увидите общие черты между вариантами использования и сможете обобщить их в более высокоуровневую утилиту. Тогда получится, что пакет `requests` собран не ради одного конкретного HTTP-запроса; он сыграет более общую роль как HTTP-клиент. Сейчас часть вашего кода может быть очень специализированной, но, обнаружив новые области, где требуется то же поведение, вы можете увидеть возможность определить роль, которую выполнит эта часть кода, немного обобщить ее и создать пакет.

Работая над переделкой своей программы для CarCorp, вы вспоминаете, что бóльшая часть кода связана с навигационной системой автомобиля. Вы понимаете, что после небольшой доработки навигационный код заработает и для транспортных средств Acme Auto*. Этот код сможет выполнять

---

* Реально существовавшая компания, производившая автомобили в штате Пенсильвания с 1903 по 1911 год. Была продана и сменила название на SGV. — *Прим. пер.*

функцию коммуникации для навигационных систем автомобилей. Поскольку вы узнали, что пакеты способны зависеть от других пакетов, и поскольку код в вашей навигационной системе уже является достаточно связным, вы решаете до следующей встречи с сотрудниками CarCorp создать не один, а два пакета.

УСПЕШНАЯ КОМПОЗИЦИЯ

Хорошие примеры композиции можно увидеть в сборке пакетов с помощью таких фреймворков Python, как Django (https://www.djangoproject.com). Фреймворк Django сам по себе является пакетом, и, поскольку он построен на основанной на подключаемых модулях архитектуре, его функциональность можно расширять, устанавливая и настраивая дополнительные пакеты. Ознакомиться с сотнями пакетов, перечисленных в Django Packages, можно на https://djangopackages.org. Это даст вам представление о широких возможностях приспособления с помощью метода сборки пакетов.

Рассуждения о композиции и декомпозиции подчеркивают тот факт, что распространение пакетов может существовать в любом объеме точно так же, как функции, классы, модули и импортируемые пакеты. Воспринимайте сцепление и связанность как маяки, помогающие соблюсти правильный баланс. Сотня дистрибутивов, каждый из которых содержит одну-единственную функцию, станет тяжким грузом для поддержания системы, а один распространяемый пакет, обеспечивающий сотню импортируемых пакетов, заработает так, словно никакого пакетирования вообще не предпринималось. В крайнем случае задайте себе вопрос: «Какую роль должен играть этот код?»

Теперь, когда вы узнали, что сборка программ в пакеты помогает писать согласованный, слабо связанный код с четкой принадлежностью, который можно предоставить потребителю доступным способом, надеюсь, вы готовы засучить рукава и погрузиться в детали.

Краткие итоги

- Пакеты архивируют программные файлы и метаданные о программном обеспечении, такие как его название, автор, тип лицензии и версия.
- Диспетчеры пакетов автоматизируют установку пакетов и управляют взаимозависимостями между ними.

- В процессе сборки пакетов вы столкнетесь с рядом подводных камней, которых можно избежать с помощью инструментов и воспроизводимых процессов.
- В репозиториях программного обеспечения хранятся опубликованные пакеты, которые могут установить другие люди.
- Сборка пакетов — прекрасный способ выделить и инкапсулировать код с высокой степенью сцепления.
- Сборка пакетов может использоваться как инструмент для ослабления связанности, чтобы добиться гибкости при разработке и сопровождении кода.
- Пакеты с версиями — отличный способ уменьшить неразбериху, возникающую в базе исходного кода при каждом отдельном обновлении.