1 Автоматизация при помощи PowerShell

В ЭТОЙ ГЛАВЕ

- ✓ Как сформулировать потребности в автоматизации
- ✓ Почему стоит использовать PowerShell
- ✓ Как определить, что PowerShell подходит для работы
- ✓ Как начать автоматизировать уже сегодня

Каждый день во всех отраслях ИТ-специалистам приходится делать больше и с меньшими затратами. Лучший способ добиться этого — автоматизация. Однако не все компании видят в ней преимущество. Нередко автоматические скрипты создаются обычными сисадминами в свободное от работы время, а следовательно, не дают существенной экономии времени и даже становятся препятствием для изменений.

Уверен, что вы уже пробовали бескодовые платформы автоматизации: IFTTT, Flow, Zapier или другие. Но если мы с вами похожи, возможности этих платформ, скорее всего, показались вам, как и мне, довольно примитивными. Они хороши для небольших персональных задач, но для решения реальных проблем уровня предприятия требуют настройки, которая невозможна при помощи их простого графического интерфейса.

И здесь на помощь придет PowerShell — фреймворк с простым, интуитивно понятным языком для написания скриптов автоматизации, который также

позволяет объединять операции в логические цепочки. При помощи *командлетов* (cmdlets) PowerShell можно выполнять операции из консолей администратора и управлять всей экосистемой Microsoft (Azure, Office 365, Microsoft Dynamics и др.) и другими платформами, в том числе Linux и Amazon Web Services. Обуздав возможности PowerShell и освоив несколько базовых принципов, ИТ-специалист может стать настоящим гуру автоматизации.

Сегодня ИТ-специалисты не только должны делать больше при меньших затратах: вся ИТ-индустрия переходит к модели «инфраструктура как код». У меня есть уникальный опыт работы в компании, которая оказывает услуги консалтинга в области инфраструктуры и создает специализированные приложения. Имея возможность работать над проектами автоматизации в рамках обеих этих сфер, я понял, что, обладая знаниями в каждой из них, можно стать настоящим экспертом.

Если вы системный администратор или другой ИТ-специалист, то, вероятно, уже умеете работать с интерфейсом командной строки (CLI), пакетными файлами и скриптами PowerShell. Поэтому шаг к написанию кода, заточенного под задачи автоматизации, будет не так уж велик. В то же время, скорее всего, у вас нет ряда полезных навыков, таких как управление исходным кодом и проведение модульных тестов. И в этом данная книга вам поможет.

С другой стороны, далеко не все опытные программисты хорошо знают тонкости системного администрирования. Однако сильная сторона PowerShell — независимость от архитектуры предприятия. Запустить скрипт на сервере так же легко, как и на локальной машине. Эта книга покажет, как можно использовать PowerShell в организациях любого размера, как создавать в ней надежные, легкие в обслуживании и безопасные системы автоматизации.

1.1. ЧТО ВЫ УЗНАЕТЕ ИЗ ЭТОЙ КНИГИ

Эта книга не только о том, как написать скрипт в PowerShell. Программированию посвящено множество ресурсов. Мы поговорим о том, как сделать PowerShell инструментом для автоматизации:

- Как автоматизировать повторяющиеся задачи.
- Как избежать распространенных ловушек при автоматизации.
- Как сопровождать скрипты и работать над ними совместно с командой.
- Как передавать скрипты конечным пользователям.

Мы изучим все эти вопросы на реальных примерах, с которыми постоянно сталкиваются ИТ-специалисты, посмотрим на код с технической и концептуальной стороны, узнаем, как его правильно структурировать. И наконец, мы научимся применять эти знания для решения актуальных задач автоматизации.

1.2. ПРАКТИЧЕСКАЯ АВТОМАТИЗАЦИЯ

Читатели этой книги, скорее всего, уже задавались вопросом: что автоматизировать? Ответ «все!», о котором подумают многие, неверен. В общем случае принято считать, что автоматизировать нужно любую повторяющуюся задачу при условии, что на ее автоматизацию уйдет меньше времени, чем на ручное выполнение. Однако не все так просто, как и многое в ИТ-индустрии. Чтобы определить, стоит ли автоматизировать какую-то задачу, нужно учесть множество факторов, а не только затраты времени.

Легко сказать, что если автоматизация требует меньше времени, чем ручная работа, задачу стоит автоматизировать. Но это еще не все. Необходимо принять во внимание следующие факторы:

- *Время:* сколько времени уходит на выполнение задачи?
- *Частота:* как часто выполняется задача?
- *Актуальность:* как долго будет нужна задача и автоматизация?
- *Реализация:* сколько времени потребуется для автоматизации задачи?
- *Обслуживание:* сколько времени потребуется на обслуживание системы и поддержание ее в рабочем состоянии?

Первые два фактора — время и частоту — обычно несложно оценить. Актуальность системы тоже достаточно очевидна. Например, если автоматизировать задачу, которая перестанет быть актуальной при следующем обновлении системы, вложенные затраты времени и сил просто не успеют окупиться.

Сложнее оценить время на реализацию и обслуживание. Этот навык придет постепенно, вместе с опытом. Помимо времени, не нужно забывать о стоимости инструментов, платформ, лицензий и т. д., а говоря об обслуживании — о затратах на поддержку этих платформ, изменения API, обновления систем.

Оценив все перечисленные факторы, можно определить, сколько времени потребуется на автоматизацию, а значит, решить, стоит ли она усилий. Эти затраты можно вычислить, умножив время на частоту и актуальность и добавив к результату время на реализацию и обслуживание. И если ручное выполнение задачи требует больших затрат, имеет смысл ее автоматизировать.

$$\text{Время} \times \text{Частота} \times \text{Актуальность} > \text{Реализация} + \\ + (\text{Обслуживание} \times \text{Актуальность}).$$

$$\text{Затраты на ручное выполнение} > \text{Затраты на автоматизацию}.$$

Неопытным специалистам трудно оценить затраты на реализацию и обслуживание. Этот навык приходит с опытом. Поэтому на данном этапе можно пользоваться простым правилом: если автоматизация сэкономит хотя бы половину времени, необходимого на ручное выполнение, скорее всего, она будет очень полезна.

Автоматизация эффективна не только для выполнения часто повторяющихся задач. Она приносит выгоду в случаях, когда процесс связан с высокой вероятностью человеческих ошибок. Отличный пример таких процессов — это работа с большими объемами данных и их преобразование. Люди часто ошибаются при наборе текста, а при работе с данными еще чаще. Поэтому все, кто когда-либо работал в Excel и использовал формулы для вычислений, уже в какой-то мере автоматизаторы.

Автоматизация может быть полезна и при решении одноразовых задач. Кроме того, если сохранить написанный скрипт, он может пригодиться для решения похожих задач в будущем. Прекрасным примером служит форматирование текста. Представим, что имеется файл с плохо отформатированным текстом, который нужно разбить на столбцы и поместить в таблицу. В нем не очень много строк, и можно выполнить эту работу вручную, потратив время на копирование и вставку. А можно заняться автоматизацией и отточить навыки применения регулярных выражений, разбиения строк, выделения подстрок, поиска и замены и многих других приемов работы со строками. Эти навыки пригодятся в будущем, при автоматизации других задач.

Еще одна область для потенциальной автоматизации — задачи, которые выполняются редко, но требуют значительных затрат времени и сил. Если процесс настолько сложен, что для его реализации необходим план, можно использовать его как основу будущего проекта: начать с автоматизации одного из этапов, затем другого, третьего и так далее, пока весь процесс не будет выполняться автоматически, нажатием одной кнопки.

Лучший способ начать — это найти простую, но часто выполняемую задачу и автоматизировать ее. Такая задача не обязательно должна быть сложной или нетипичной. Достаточно подумать о том, на чем можно сберечь время.

Автоматизация может помочь более продуктивно решать повседневные задачи, отвлекающие от основной работы. Например, я не всегда успеваю разбирать почту: прочитываю сообщения, но они остаются в папке «Входящие» и копятся там в больших количествах. Для этой цели можно использовать правила Outlook, но с ними я не уверен, что не пропущу важное оповещение, особенно когда нахожусь не на рабочем месте. Поэтому я написал скрипт, который делает все за меня: перемещает письма в определенные папки по электронным адресам или ключевым словам. Благодаря этому я не только экономлю время, но и работаю более продуктивно и получаю удовлетворение от того, что моя почта остается в порядке.

Наконец, следует помнить: весь процесс до мелочей автоматизировать не обязательно. Например, если расчеты показывают, что на полную автоматизацию уйдет больше времени, чем на ручное выполнение задачи, можно автоматизировать ее частично. Отличный пример — штрихкоды. Они позволяют кассирам и кладовщикам быстро сканировать товары и не вводить артикулы вручную. Метки RFID могут сделать эту работу еще эффективнее, но их внедрение обойдется дороже.

По мере накопления опыта вы сможете лучше понимать, что стоит, а что не стоит автоматизировать. И как мы увидим в следующем разделе, поэтапный подход к работе и применение готовых блоков из ранее написанных скриптов заметно упрощают создание сложных, качественных систем автоматизации.

Для начала рассмотрим четыре ключевых фактора, которые необходимо учитывать при планировании автоматизации. К ним относятся:

- цель;
- триггеры;
- действия;
- обслуживание.

Цель автоматизации — это стоящая перед программистом задача. *Триггеры* запускают *действия*, или этапы процесса автоматизации. Наконец, *обслуживание* — это мероприятия, которые необходимы для поддержания работоспособности всей системы автоматизации либо отдельных ее частей.

Чтобы подробно поговорить об этих факторах, рассмотрим реальный пример. Представим веб-сервер, который периодически прекращает работу из-за того, что логи заполняют все место на диске. Логи нельзя удалять. Они нужны для проверок безопасности. Поэтому где-то раз в неделю нужно архивировать старые, созданные более месяца назад логи и помещать архивы в долговременное хранилище.

1.2.1. Цель автоматизации

Цель автоматизации — это задача, которую требуется решить путем автоматизации. Может показаться, что определить цель очень просто. Однако необходимо точно учесть все, что предстоит сделать.

В примере с архивацией логов цель — предотвратить переполнение дисков на сервере. Но это лишь небольшая видимая часть задачи. Если бы цель состояла только в этом, достаточно было бы просто удалять старые логи. Однако они нужны для проверок безопасности. Поэтому требуется написать скрипт, который не только освободит место, но и не допустит потери данных и при необходимости обеспечит к ним доступ. Исходя из этого можно представить процесс автоматизации и составить список действий, которые должны выполняться.

Если, например, понадобится периодический доступ к логам, список действий будет другим, поскольку архивация и долговременное хранилище будут не самым лучшим решением. Вместо этого файлы придется перемещать в накопитель бóльшего объема. Тогда диск не будет переполняться, а логи будут легкодоступны. Теперь, когда мы узнали, какие задачи стоят перед системой автоматизации, составим план их выполнения.

1.2.2. Триггеры

Триггеры приводят систему автоматизации в действие. В общем случае они бывают двух типов: на основе опроса и на основе события. Триггер на основе опроса срабатывает при наступлении определенных условий, а триггер на основе события — когда происходит внешнее событие. Выбор триггера существенно влияет на процесс автоматизации.

Триггеры на основе опроса регулярно проверяют систему на выполнение заданных условий. В этой книге мы будем говорить о двух самых распространенных типах таких триггеров: *наблюдателях* и *расписаниях*.

Триггер-наблюдатель ожидает выполнения какого-то условия. Это может быть что угодно: от появления новых файлов на FTP-сервере до поступления электронных сообщений или подтверждения запуска сервиса. Такие триггеры могут работать непрерывно или с определенной периодичностью.

Выбор периодичности проверки зависит от потребностей системы и затрат ресурсов. Например, если ведется наблюдение за поступлением новых файлов и хорошо известно, что они появляются примерно раз в час, проводить проверку каждые 60 секунд нерационально.

Триггер-расписание также работает с определенной периодичностью, но при этом никаких условий не проверяет. Вместо этого каждый раз запускается скрипт, выполняющий все необходимые действия. Типовые примеры таких систем: очистка файлов, синхронизация данных и периодическое обслуживание системы. Как и в предыдущем случае, нужно подобрать достаточный интервал между запусками.

Триггер на основе события срабатывает при наступлении определенного внешнего события, например поступлении HTTP-запроса, такого как веб-хук. Триггеры событий также могут включать вызовы от других средств автоматизации, и большинство инструментов службы поддержки предусматривают рабочий механизм, который может запускать автоматизацию при получении определенного запроса. Подобные механизмы часто используются в системах технической поддержки. Это всего несколько примеров триггеров на основе событий, но таким триггером можно считать любое внешнее взаимодействие.

Простое нажатие кнопки или выполнение команды в терминале может быть триггером на основе события. Важно помнить, что триггеры на основе события инициируются внешними событиями, а триггеры на основе опроса — выполнением определенных условий.

Вернемся к нашему примеру с архивацией логов. Необходимо решить, триггер какого типа использовать. В данном случае лучше всего подойдет триггер на основе опроса, поскольку веб-сервер не выдает никаких оповещений при записи логов. Триггеры-наблюдатели обычно нужны при решении задач, требующих

немедленной реакции, например при перезапуске служб после сбоя или восстановлении сетевого соединения. Удаление логов веб-сервера — это периодическое обслуживание, поэтому в данном случае лучше использовать триггер-расписание.

Определимся с периодичностью запуска. Как мы уже знаем, логи нужно удалять не реже одного раза в неделю. Поэтому логично настроить триггер на интервал, не превышающий неделю. Новый лог создается, когда предыдущий достигает определенного объема. Представим, что в день появляется три-четыре лога. Это означает, что оптимальным интервалом для удаления будут сутки. При более частых запусках системы логов может стать слишком много, а более редкий запуск опасен. Определив триггер, перейдем к основной части системы автоматизации: действиям.

1.2.3. Действия

Действия — это то, что чаще всего и понимается под автоматизацией. Это операции, которые выполняет система автоматизации для достижения поставленной цели. В рамках одной системы может осуществляться множество действий (*этапов*). Действия можно разделить на три категории: *логика*, *задачи* и *логирование*. Действия, которые нужно выполнить для автоматизации удаления логов, показаны на рис. 1.1.

Логика управляет порядком работы системы. К ней можно отнести условные конструкции (например, if/else), циклы, ожидания, перехват/обработку ошибок, анализ переменных и других данных. Задачи — это действия, выполняемые при выполнении определенных условий, то есть все, что не относится к логике или регистрации. Можно сказать, что логика — это мозг системы, а задачи — ее руки.

Логирование, как следует из самого слова, — это запись выполненных действий. Журналы логов содержат записи о результатах выполнения логики и задач. Хотя логирование фактически является задачей, лучше рассматривать ее отдельно, так как сама по себе она не служит цели автоматизации. Тем не менее логирование необходимо в любой успешной и поддерживаемой системе автоматизации.

Рассматривая наш пример, можно определить состав и типы действий, которые нужно выполнить:

1. Поиск логов старше 30 дней (логика).
2. Создание файла архива, имя которого содержит метку времени (задача).
3. Добавление старых логов в архив (задача).
4. Удаление старых логов с диска (задача).
5. Запись удаленных логов и имени файла архива (логирование).
6. Загрузка файла архива в хранилище Azure Blob для долгосрочного хранения (задача).

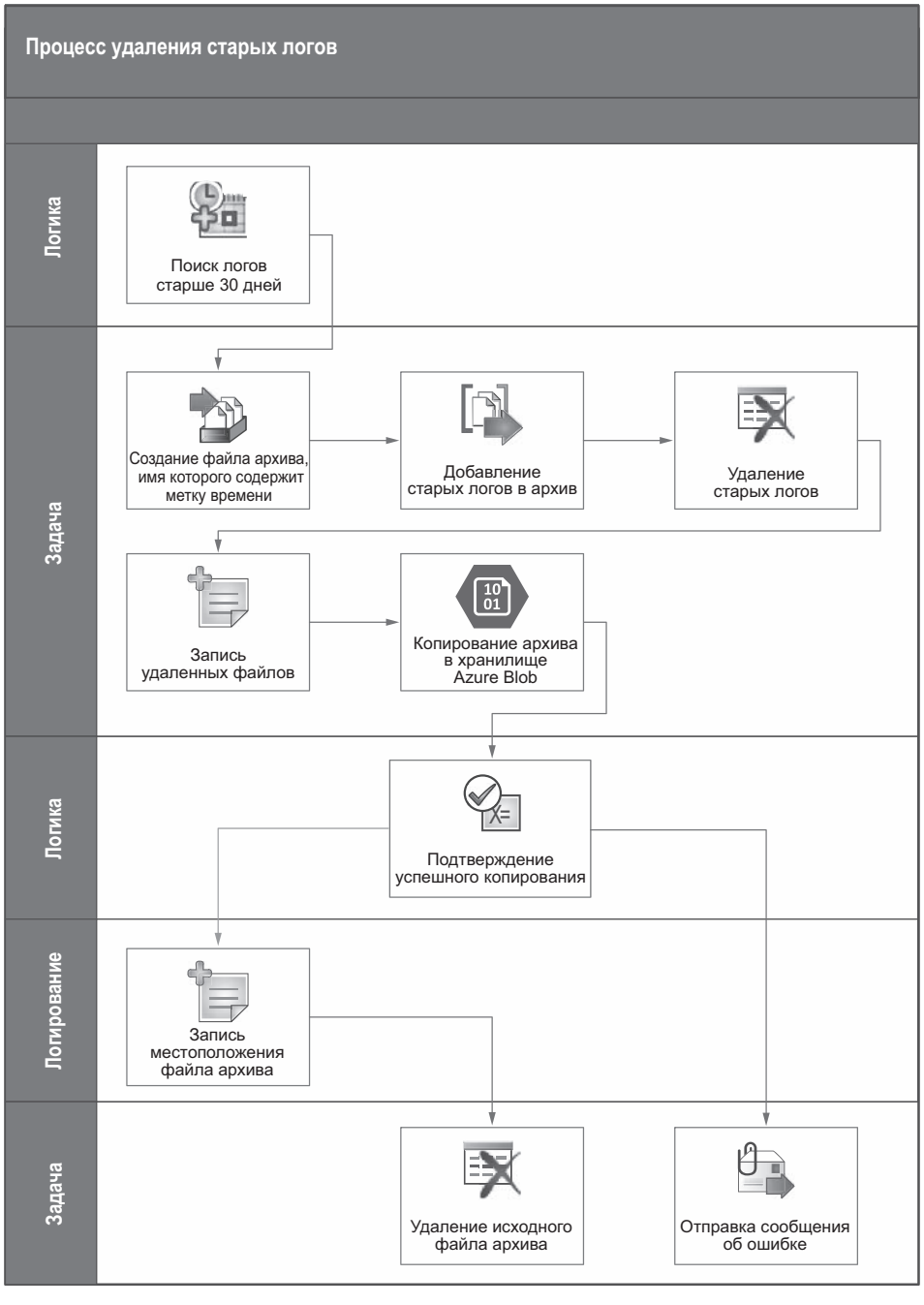

Рис. 1.1. Этапы автоматизации удаления логов и их классификация

7. Проверка результатов загрузки (логика). В случае ошибки — завершение работы и отправка уведомления.
8. Запись данных о местоположении архива (логирование).
9. Удаление архивного файла из локальной файловой системы (задача).

1.2.4. Обслуживание

Несколько лет назад я помогал одному заказчику автоматизировать процесс создания профилей пользователей. При обсуждении этой задачи мне сказали, что профили пользователей должны сначала поступать в общую папку, находиться там около часа, а затем перемещаться в папки по группам. Как оказалось, это необходимо для синхронизации с внутренним приложением. Выяснилось, что специалист, который должен был добавлять пользователей в это приложение, автоматизировал свою работу и сэкономил от 30 до 60 минут в неделю.

В то время все пользователи были в одной, общей группе. Однако со временем все изменилось: компания расширилась, появились группы пользователей, которые теперь размещались в отдельных папках. Через какое-то время обнаружилось, что новые пользователи не регистрируются во внутреннем приложении. Но разработчик автоматизации давно ушел из компании, а кроме него никто не знал, как она работает. Поэтому пользователей сначала добавляли в общую папку, ждали синхронизации с приложением (которая выполнялась раз в час), а затем распределяли пользователей по другим папкам. То, что раньше экономило 60 минут одному человеку, теперь обходилось его коллегам в несколько дополнительных минут на каждого нового пользователя. Со временем автоматизация стала тратить времени больше, чем экономила.

Этот пример показывает, что случается, когда никто не думает о будущем. Никто не может предвидеть будущее, но кое-что можно предусмотреть. На любом этапе процесса нужно спрашивать себя: насколько сложным будет обслуживание этого процесса. И, опираясь на опыт работы, продумывать возможные изменения требований.

В рассматриваемом нами примере действия начинаются с поиска старых логов, созданных 30 и более дней назад. Первое, о чем нужно подумать, — не будет ли диск заполняться быстрее. Тогда удалять логи придется каждые две недели. Насколько сложно будет внести такое изменение? Если для отбора по времени используется переменная — совсем не сложно. Если же цифра жестко закодирована, может потребоваться много работы.

Другой возможный сценарий развития системы — появление еще одной папки с логами. Насколько это вероятно? Если вероятность велика, нужно сразу подумать о том, чтобы скрипт умел работать с набором папок. А если она мала, можно запускать программу несколько раз, для каждой отдельной папки.

Нелишне подумать и о возможном изменении периодичности запусков. Например, может потребоваться выполнять скрипт не один раз в день, а один раз в час. Может показаться, что сделать это несложно: достаточно изменить интервал времени в фильтре. Однако необходимо учесть влияние этого изменения на другие процессы. Например, если при создании файла архива к его имени добавляется метка времени, она должна иметь значение с точностью до часов. Иначе часть логов может быть потеряна в результате перезаписи архива.

Ответы на любой из этих вопросов зависят от конкретных потребностей. Нельзя предвидеть все возможные ситуации, но если задумываться о них, а также об их реализации в PowerShell, можно заранее подготовиться к изменениям и быстро внести их по мере необходимости.

Однако не следует слишком увлекаться анализом будущих изменений и оценкой их вероятности. Так можно потратить уйму времени и ничего не создать или же сделать логику системы настолько запутанной, что в ней будет сложно разобраться. Необходимо чувствовать баланс и понимать, где нужно остановиться. Об этом мы еще не раз здесь поговорим.

1.3. ПРОЦЕСС АВТОМАТИЗАЦИИ

В начале работы над проектом автоматизации существует риск взвалить на себя непосильное бремя. Часто приходится слышать о принципе «не усложняй» (KISS), но придерживаться его на практике не всегда получается, особенно когда необходимо работать с несколькими связанными системами, использовать сложную логику, да еще и при постоянно меняющихся требованиях. На помощь приходят концепции *стандартных блоков* и *этапов*, которые позволяют разбить сложные задачи на небольшие, простые этапы.

1.3.1. Стандартные блоки

Какой бы сложной ни была программа, ее можно разбить на небольшие простые этапы или стандартные блоки. Такой подход помогает снизить нагрузку, уточнить цели и достигать их регулярно. Кроме того, при поэтапной разработке отдельные части планируемой системы можно будет использовать по мере их создания и на их основе создавать дальнейшие блоки. Большая часть этой книги будет посвящена созданию различных стандартных блоков, которые можно по мере необходимости использовать в разных проектах.

Стандартные блоки также позволяют развивать навыки работы: с их помощью вы станете лучше разбираться не только в написании кода, но и во всем процессе. С каждым новым проектом навыки будут совершенствоваться. Если вы используете стандартные блоки и нашли более удачное решение задачи, вы можете вернуться к старым проектам и обновить их быстро и без усилий.

1.3.2. Этапы

Существует поговорка: «Прежде чем начать бегать, нужно научиться ходить». Она идеально подходит к автоматизации. Первые несколько скриптов, которые вы напишете, вряд ли будут хороши: совсем как первый рисунок или первое школьное сочинение. Для развития навыков нужны время и опыт. Но это не означает, что ваши первые программы будут бесполезны.

Разбив проект на этапы, можно двигаться к цели постепенно. Представьте себе, что нужно добраться из раздела А в раздел Б. Конечно, лучше всего сесть на автомобиль. Но что поделать, если вы не знаете, как его собрать? Поэтому начните с малого и двигайтесь вперед: сначала это будет скейтборд, потом скутер, велосипед, мотоцикл и, наконец, автомобиль. Преимущества такого подхода показаны на рис. 1.2. На каждом этапе вы будете вносить улучшения и развивать свою программу. Более того, она будет приносить пользу с самого начала. Если же сразу взяться за автомобиль, придется ходить пешком до завершения работы.

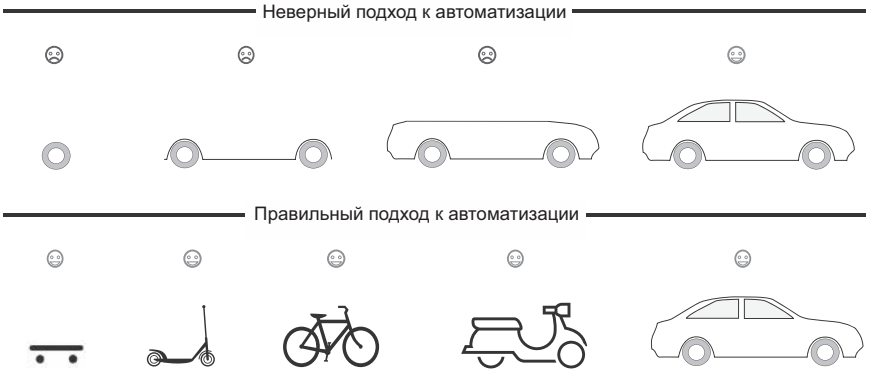

Рис. 1.2. Поэтапный подход к работе позволяет получать выгоду с самых первых этапов

На каждом этапе работы вы, вероятно, создадите несколько стандартных блоков, которые можно использовать на разных этапах, а также постепенно усовершенствовать. Например, на первом этапе рис. 1.2 у вас получилось колесо. Далее на этапе 2, углубив знания, вы можете его улучшить.

Разбивка на этапы также помогает адаптировать проект по ходу работы. После каждого этапа можно собирать обратную связь, чтобы обнаруживать невыявленные проблемы. Если продолжить пример рис. 1.2 и остановиться на скейтборде, из обратной связи можно узнать, что скейтборд не подходит для поездки по грязным дорогам. Можно учесть это замечание и добавить на этапе 2 действие для увеличения колес. Если же выполнять проект целиком, без выделения этапов, может оказаться, что готовый автомобиль застревает в грязи, а другие колеса к нему не подходят, так как на них не рассчитана подвеска. Придется переделать очень много деталей.

1.3.3. Сочетание стандартных блоков и этапов

Чтобы продемонстрировать концепцию стандартных блоков и этапов на более характерном для ИТ примере, можно поговорить об автоматизации создания виртуальных машин. Этот довольно сложный процесс можно разбить на несколько этапов:

1. Создание виртуальной машины.
2. Установка операционной системы.
3. Настройка операционной системы.

Было бы хорошо написать такой скрипт целиком, за один прием. Но это большая работа, эффект от которой будет виден лишь после завершения. Поэтому лучше работать над каждым этапом в отдельности и сразу пользоваться результатами. Начнем с этапа 1 — создания виртуальной машины. В нем можно выделить несколько стандартных блоков:

1. Выбор хост-компьютера.
2. Создание пустой виртуальной машины.
3. Выделение памяти и мощности процессора.
4. Подключение сетевой карты к соответствующей подсети.
5. Создание и подключение виртуальных жестких дисков.

По завершении этапа 1 (создание виртуальной машины, рис. 1.3) можно перейти к этапу 2, а полученные результаты использовать для продолжения работы.

На этапе 2 необходимо установить операционную систему. Эту задачу можно решить несколькими способами. Можно создать шаблон виртуального жесткого диска с уже установленной операционной системой. Однако такой шаблон придется поддерживать, в том числе устанавливать обновления, а также возиться с синхронизацией, если виртуальные машины будут работать в разных регионах. Поэтому лучше использовать средства управления конфигурациями. При этом образ всегда будет синхронизирован и обновлен до последней версии.

На этом этапе становится ясно, что для получения образа виртуальная машина должна быть подключена к определенной подсети. Поэтому набор стандартных блоков будет таким:

1. Подключение к подсети для развертывания операционной системы.
2. Включение виртуальной машины.
3. Ожидание установки операционной системы.
4. Подключение к рабочей подсети.

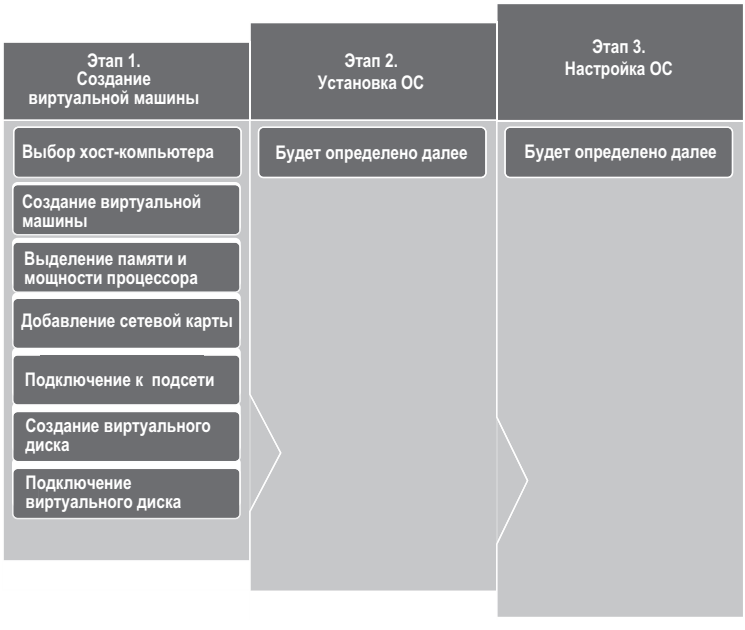

Рис. 1.3. Поэтапный подход к созданию и настройке виртуальных машин (этап 1)

Созданный на этапе 1 блок для подключения ВМ к подсети можно использовать и на этом этапе (действия 1 и 4). Я не случайно выделил подключение к подсети в отдельный блок: мне уже приходилось решать похожие задачи, и я не раз сталкивался с аналогичной ситуацией. Если объединить все операции, то есть настройку процессора, памяти, сетевой карты и виртуального жесткого диска в один блок, его нельзя будет использовать на других этапах. Повторная настройка процессора и памяти хоть и будет напрасной тратой времени, пройдет без последствий. А вот подключение уже подключенного диска приведет к ошибке. Если вы не учтете этот момент и решите создать единый блок для всех операций, ничего страшного. В любом случае этот опыт будет полезен. Я до сих пор иногда совершаю такие промахи. К тому же, написав отдельный стандартный блок для подключения к подсети, можно вернуться на прошлый этап и обновить соответствующий код.

Развитие проекта на этапе 2 показано на рис. 1.4.

Итак, в работе уже два этапа, и их результаты видны пользователям. Можно собрать от них обратную связь и выяснить, какие настройки включить в план реализации на этапе 3 (рис. 1.5). Например, можно назначить статический IP-адрес, подключить дополнительные диски или выполнить другие действия, которые ранее не планировались. Этап 3 не обязательно должен быть последним. Например, можно предложить этап 4, на котором будут автоматически устанавливаться приложения.