Оглавление

Об авторе

Гэри Грувер (Gary Gruver) – руководитель с выдающимся опытом преобразования процессов разработки и использования программного обеспечения в крупных организациях. Гэри работал в должности директора по исследованиям и развитию подразделения встроенного программного обеспечения HP LaserJet, в которой полностью изменил процесс разработки «прошивок», а затем в качестве вице-президента по релизам и эксплуатации в Macy's.com, где он реализовал процесс непрерывной поставки. В настоящее время он консультирует крупные организации и проводит семинары, чтобы помочь им трансформировать процессы разработки и поставки программного обеспечения. Он является соавтором книг *«Возглавляя трансформацию: применение гибких и DevOps-принципов в большом масштабе»* и *«Практический подход к крупномасштабной гибкой разработке: как компания HP изменила встроенное ПО LaserJet»*.

Веб-сайт: GaryGruver.com
Twitter: @GRUVERGary
Linkedin: https://www.linkedin.com/in/garygruver
Электронная почта: gary@garygruver.com

Благодарности

Многие люди внесли свой вклад в эту книгу. Я хотел бы поблагодарить всех, с кем я работал многие годы, кто помог мне лучше понять, как разрабатывать программное обеспечение. Идеи, которыми я делюсь в данной книге, являются накоплением всего, что я узнал, работая с каждым из вас в постоянном путешествии по совершенствованию процессов разработки. Без наших обсуждений и дискуссий моё понимание не было бы таким богатым, а книга не была бы такой полной.

Я хотел бы особо поблагодарить всех клиентов моих семинаров для руководителей, которые позволили мне принять участие в вашем путешествии. Обсуждение проблем, с которыми вы столкнулись, а также улучшений, которые сработали, помогли мне улучшить содержание этой книги. Спасибо также **Полу Ремейсу** (Paul Remeis) и **Грегу Лоннону** (Greg Lonnon) за помощь в создании и проведении семинаров.

Кроме того, я хотел бы поблагодарить всех, кто нашёл время, чтобы дать мне обратную связь о ранних версиях книги (в алфавитном порядке): **Джону Эдигеру** (John Ediger), **Мирко Герингу** (Mirco Hering), **Джезу Хамблу** (Jez Humble), **Томми Маузеру** (Tommy Mouser) и **Видон Перис** (Vidon Peris). Ваш вклад значительно улучшил конечный продукт.

Я также хотел бы поблагодарить редакционный и производственный персонал: редактор **Кейт Сейдж** (Kate Sage) своей настойчивостью очень помог мне прояснить идеи, чтобы они были чётко и лаконично переданы. Постоянные итерации сделали книгу лучше, но, что более важно, заставили меня выкристаллизовать концепции, позволяющие более эффективно помочь другим в их путешествиях. **Шала Мадави** (Shahla Mahdavi) и **Кэсси Лайдон** (Cassie Lydon) из бюро дизайна «Katie Bush» обеспечили большую часть иллюстраций. Они отлично справились с созданием визуальных артефактов, помогающих передать идеи, о которых я говорю в книге. Наконец, я хотел бы поблагодарить **Амелию Тидеманн** (Amelia Tiedemann) за прекрасные фотографии слонов и дизайн обложки[1]. Мне кажется, что такая обложка очень помогает передать важное сообщение: успешная DevOps-трансформация требует больше, чем просто наличия всех необходимых составных частей.

[1] Речь про обложку оригинального издания. – *Прим. перев.*

Предисловие

Когда мы с Дэвидом Фарли (David Farley) писали книгу «Непрерывная поставка», мы считали, что занимаемся пыльным, нишевым уголком жизненного цикла программного обеспечения. Мы не ожидали большого интереса к книге, просто нам было больно видеть, как люди тратят недели на развёртывание сборок в тестовые среды, выполняя в основном ручные регрессионные тесты, занимающие недели и месяцы, проводя за этим занятием свои ночи и выходные, борясь с возникающими сбоями и простоями. Мы знали, что многие процессы разработки программного обеспечения работали крайне неэффективно и давали плохие результаты с точки зрения качества и стабильности производимых систем. Из нашего опыта работы на крупных предприятиях мы знали, что существуют инструменты и практики, позволяющие устранить многие из этих проблем, если бы только команды систематически применяли их.

К счастью, мы были не единственными, кто это видел. Многие другие по всему миру, включая Гэри, пришли к такому же выводу; так родилось движение DevOps. Движение имело беспрецедентный успех прежде всего потому, что эти идеи работают. Поскольку я работал с первыми лицами в крупных регулируемых компаниях, а в последнее время в качестве сотрудника федерального правительства США в Департаменте 18F, я видел, насколько радикально улучшались сроки поставки ПО, сопровождаемые улучшением качества и устойчивости, даже при работе со сложными унаследованными системами.

Самое главное, я видел, что эти идеи приводят к более счастливым ИТ-сотрудникам и конечным пользователям. Используя непрерывную поставку, мы можем создавать продукты, успех которых связан с коллективным подходом к разработке продукта, поощряющим экспериментирование. Все в команде вносят свой вклад в изучение того, как создавать лучшие результаты для пользователей и организаций. Конечные пользователи получают огромную выгоду, если мы можем работать с ними с самого начала процесса разработки, итеративно, быстро, меняя дизайн систем в ответ на их отзывы и предоставляя наиболее важные функции с самого начала жизненного цикла продукта.

Гэри начал применять идеи DevOps и непрерывной поставки ещё до того, как эти слова стали популярными, начиная с его работы в HP, где он возглавлял команду разработки встроенного программного обеспечения LaserJet. Его большая распределённая команда применяла принципы непрерывной поставки к «прошивкам» печатающих устройств и показала впечатляющие результаты качества и производительности в области, где никто не заботился о частых релизах. Затем он продолжил делать то же самое в регулируемой организации со сложными монолитными унаследованными системами.

Сегодняшние технологические лидеры понимают необходимость трансформации своих организаций для достижения одновременно как лучшего качества, так и более высокой производительности. Эффективное лидерство имеет важное зна-

чение для успешности преобразований. Однако преодоление таких препятствий, как организационная инерция, изолированное мышление и высокая архитектурная сложность, может показаться неподъёмной задачей. В этой книге представлено краткое, но подробное руководство для лидеров по инженерным практикам и архитектурным изменениям, которые имеют решающее значение для достижения прорывных результатов.

Эта книга не сделает ваше путешествие лёгким, но она послужит бесценной картой, ведущей по верной дороге. Счастливого пути!

Джез Хамбл (Jez Humble)

Глава 1:

DevOps и конвейер развёртывания

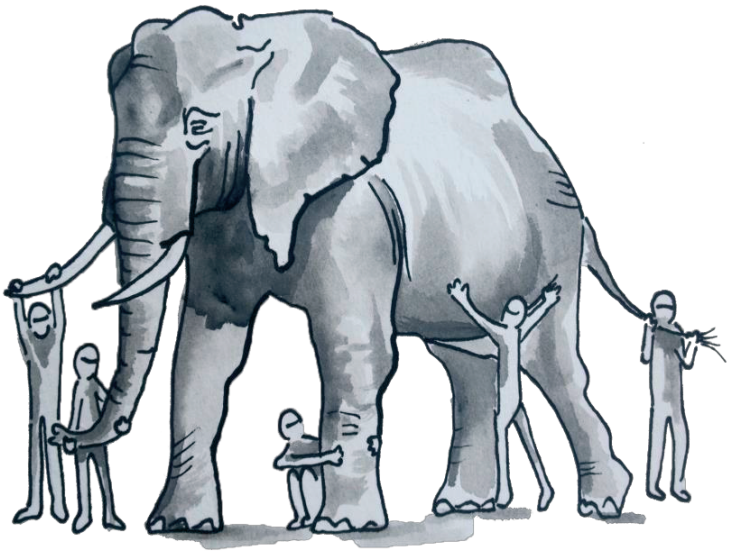
Программное обеспечение играет всё большую роль в том, как компании конкурируют в самых разных отраслях экономики. При смещении конкурентных преимуществ в сторону программного обеспечения крупные традиционные организации осознают, что их нынешние подходы к управлению ПО ограничивают способность компании реагировать с той скоростью, которая требуется основному бизнесу. DevOps – основательное изменение в практике управления программным обеспечением и работой ИТ-подразделения. Это своеобразный ответ на требования динамичных компаний более быстро выполнять изменения в бизнесе, а также результат осознания того, что крупные разработчики программного обеспечения применяют принципы DevOps для разработки нового программного обеспечения быстрее, чем кто-либо мог подумать. Все говорят о DevOps.

По своему роду занятий я встречаюсь со множеством совершенно разных компаний. Я быстро понял, что DevOps означает разные вещи для разных людей. Все они хотят *делать* DevOps ради всех тех преимуществ, о которых слышат, но они не совсем точно знают, что же такое DevOps, с чего начать или как управлять улучшениями с течением времени. Они слышат множество отличных идей о DevOps, но испытывают серьёзные затруднения в поиске единого верного определения DevOps и формулирования списка необходимых изменений. Всё это похоже на пятерых слепых, описывающих слона. В крупных организациях отсутствие единого понимания подхода DevOps препятствует прогрессу и приводит к размытию фокуса. Эта книга призвана помочь структурировать и согласовать улучшения, предоставив фундамент, который могут использовать крупные организации и их руководители, чтобы понять принципы DevOps в контексте их текущих процессов разработки ПО.

Часть проблемы применения принципов и практик DevOps заключается в том, что существует слишком много идей о том, что такое DevOps, слишком много разных определений. Наиболее последовательное и всестороннее определение, которое я слышал в последнее время, предложил Джин Ким (Jene Kim), соавтор книг *«Проект Феникс»* и *«Руководство по DevOps»*. Он – великий мыслитель и евангелист движения DevOps. Чтобы задать нам общую систему координат, мы будем использовать его определение DevOps:

> *DevOps должен определяться результатами. Это тот самый набор культурных норм и технологических практик, который помогает обеспечить*

быстрый поток плановой работы, в числе прочего, из разработки через тестирование в эксплуатацию, обеспечивая при этом надёжность, исполнение и безопасность мирового класса. DevOps не о том, что вы делаете, а о том, каковы ваши результаты. Очень много понятий, которые мы связываем с DevOps, такие как культура и коммуникации, попадают под этот довольно широкий спектр убеждений и практик.



У людей есть разные взгляды на DevOps, так как то, что необходимо для улучшения качества и потока на каждом шагу, от бизнес-идеи до рабочего кода у клиента, отличается для разных организаций. Принципы DevOps, предназначенные для совершенствования процесса, во многом сводятся к реализации изменений, помогающих координировать работу между командами. Движение DevOps началось с передовых рубежей, довольно небольших компаний, которые выдавали рабочий код чаще, чем считалось возможным. DevOps также стал очень успешным в таких крупных организациях, как Amazon, где пришлось кардинально переделать монолитные системы, чтобы позволить небольшим командам работать независимо. Совсем недавно DevOps начал использоваться в крупных организациях с жёсткими архитектурными решениями, требующими координации работы сотен людей. Проблема масштабирования DevOps на более крупные организации заключается в том, что люди предполагают, что подходы к успешной координации работ в небольших группах будут такими же и для крупных организаций. Реальность такова, что хотя основные принципы одинаковы и для малых, и для больших, реализации могут и должны различаться.

Многие крупные организации не имеют общего контекста, когда они начинают своё путешествие в DevOps. Разные люди разных должностей посещают разные конференции, чтобы узнать о DevOps из презентаций компаний разного масштаба и с разными проблемами, возвращаясь с разными взглядами на то, что же такое DevOps. Каждый участник даёт очень точное описание своей «части слона», но общего понимания не возникает. Поэтому, когда они собираются создать своего собственного «слона», никто не может договориться о том, с чего же начать, а многие предлагают к реализации идеи, хорошо работающие для небольших команд, но совершенно не предназначенные для крупных организаций, требующих координации работы сотен людей. Цель этой книги – предоставить общий взгляд на «слона», чтобы помочь крупным организациям сформировать единое представление и дать структуру, которую можно использовать для определения точки начала и построения постепенного улучшения разработки программного обеспечения с течением времени.

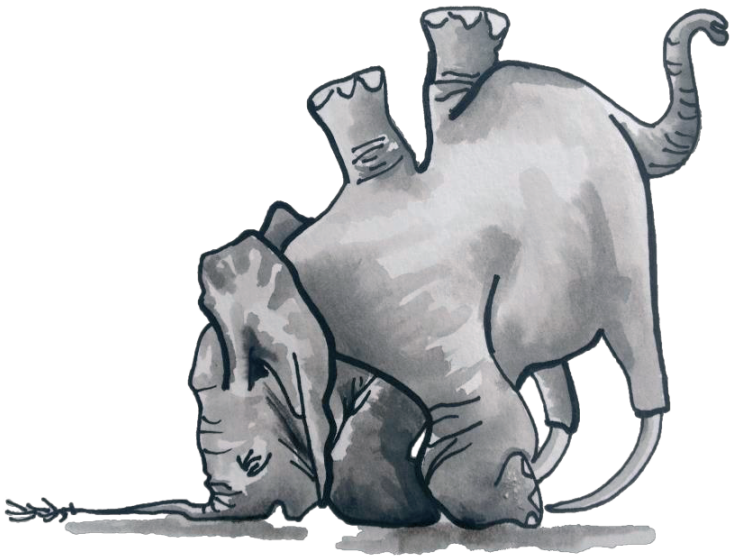
Если вы не сможете в крупной организации выстроить общее понимание персонала относительного того, что мы собираемся построить, и того, какой подход мы будем использовать для определения приоритетов в совершенствовании, то ожидать значительных результатов трансформации не следует. Возможно, отдельные элементы DevOps, о которых слышали сотрудники, будут реализованы, но существенного сдвига в скорости поставки программного обеспечения с соблюдением и улучшением всех аспектов качества не появится. Пятеро слепых могут попробовать «построить слона» из отдельных, им понятных частей, но этот «слон» не будет выглядеть и вести себя как настоящий слон, потому что исполнители зачастую не в состоянии увидеть общую картину.

Чтобы прояснить комплексный взгляд на DevOps, мы рассмотрим, как бизнес-идея проходит путь через разработку (написание программного кода), подготовку окружения, сборку и тестирование, затем передаётся в среду эксплуатации, где функционирование результата отслеживается системами мониторинга. Процесс преобразования бизнес-идеи в результат, ценный для конечного клиента, через так называемый «конвейер развёртывания»[1], был первоначально описан Джезом Хамблом и Дэвидом Фарли в их книге *«Непрерывная поставка»*. Настоящая книга будет постоянно опираться на эту концепцию, потому что я считаю, что она представляет собой фундамент DevOps. Указанная концепция описывает движение потока бизнес-идей до конечного пользователя, а также точки контроля, необходимые для поддержания или улучшения качества.

Мой личный опыт свидетельствует, что создание, документирование, автоматизация и оптимизация конвейера развёртывания в крупных компаниях, разрабатывающих программное обеспечение, а также в ИТ-организациях являются ключом к повышению их эффективности и рациональности. Как правило, у вас уже есть то, что используется для «перемещения» программного кода через организационную структуру от идеи до эксплуатации, – это и есть ваш конвейер развёртывания. Но его документирование с целью создания единого, общего представления и получения возможности оптимизации, в том числе цепочки создания ценности, является ключевым инструментом подготовки организации к внедрению DevOps. Конвейер развёртывания определяет и документирует по-

[1] DP, Deployment pipeline.

ток программного кода через систему разработки, а его сопоставление с цепочкой создания ценности помогает выявлять узкие места, потери и другие недостатки, которые могут быть устранены с использованием практик DevOps. Улучшение конвейера потребует больших организационных изменений, но единый конвейер поможет каждому понять, какие именно процессы меняются в каждый момент времени и как именно они должны измениться.



Для крупной организации с жёсткой архитектурой конвейер развёртывания – довольно сложная концепция. Поэтому в главе 2 мы начнём рассмотрение с простейшего примера конвейера, состоящего из одного программиста, и продемонстрируем примеры неэффективности, возникающие в таком простом случае. Затем в главе 3 мы остановимся на подходах DevOps, которые могут применяться для решения выявленных проблем. Мы также обсудим метрики, которые вы можете начать собирать и анализировать, чтобы понять масштабы неэффективности и для определения приоритетов в устранении проблем, которые принесут наибольшую пользу.

Как только базовая конструкция конвейера развёртывания будет хорошо разобрана, в главе 4 мы покажем, как изменяется сложность при масштабировании от одного программиста до команды разработчиков. Обеспечение совместной работы над приложением с одновременным поддержанием ветви разработки в состоянии постоянной готовности к развёртыванию является фундаментальным сдвигом для большинства традиционных организаций. Потребуются не только новые технические решения, но и культурные изменения, в частности приоритет

сохранения стабильности программного кода над разработкой новой функциональности. Это будет большим, но крайне важным сдвигом для большинства организаций, потому что если вы не можете приучить программистов быстро реагировать на обратную связь, создающуюся в конвейере развёртывания, то польза от его наличия будет очень небольшой.

После освоения практик DevOps на уровне команды крупные организации сталкиваются со следующим вызовом – потребностью в масштабировании DevOps на всю компанию. Обычно используется прямое тиражирование освоенных методов на другие подразделения, ведь эти методы уже успели себя положительно зарекомендовать. Такой подход не учитывает того факта, что самые большие препятствия не являются техническими, но связаны с управлением организационными преобразованиями и изменениями в работе персонала. Ключом к успешному масштабированию является применение в рамках всей организации единых базовых принципов с одновременным предоставлением достаточной автономности при формировании рабочих планов. Чтобы показать, как соблюсти баланс единых принципов и гибкости на местах, в главе 5 мы рассмотрим, как сегментировать работу крупных организаций на максимально мелкие участки, что позволяет более эффективно распределять ответственность и делегировать функции контроля. Для некоторых компаний, применяющих модульную архитектуру, такой способ приведёт к появлению множества небольших независимых команд, где основной задачей станет координация работы нескольких десятков людей. Для других организаций с жёсткой архитектурой, выполняющих разработку, тестирование и релиз больших приложений, потребуется взаимная увязка работы уже сотен людей. Важно начать с группировки приложений по приведённым выше категориям, так как способы координации нескольких десятков сотрудников разительно отличаются от способов координации сотен людей. Небольшие команды всегда будут более эффективными и выпускать релизы чаще, однако документирование, автоматизация и постоянное совершенствование конвейера развёртывания гораздо важнее для координации работы сотен людей, поскольку неэффективность в крупных организациях часто выражена в гораздо большей степени.

В главе 6 мы кратко рассмотрим подходы, которые дают хорошие результаты для крупных организаций с небольшими командами, работающими самостоятельно. Эта тема не будет излагаться подробно, потому что многие другие публикации о DevOps уже неплохо её раскрывают. В главе 7 мы приступим к рассмотрению сложностей проектирования конвейера развёртывания для больших, тесно связанных систем. Мы покажем, как разбить проблему на более мелкие, управляемые части, а затем собрать их вместе в комплексные системы. В главе 8 мы рассмотрим, как начать оптимизацию этих комплексных конвейеров, в том числе разработать необходимые метрики, чтобы помочь сфокусировать изменения в тех областях, где они будут наилучшим образом способствовать работе конвейера. В главе 9 мы рассмотрим и обсудим различия между применением улучшений в небольших независимых командах и в крупных сложных системах.

Преобразование работы большой организации занимает долгое время, так как требуется изменить и образ мышления, и подходы к исполнению работы. При рассмотрении такого типа организационных изменений важно учитывать ряд факторов. Во-первых, начинайте с областей, где отдача будет наиболее заметной,

чтобы задать положительный импульс. Во-вторых, найдите и задействуйте высших руководителей, которые готовы возглавить изменения и определить приоритеты улучшений: необходимо оптимизировать весь конвейер развёртывания, а не позволять отдельным группам улучшать только свой сегмент конвейера.

Построенный и работающий конвейер развёртывания обеспечивает очень хорошую основу для трансформации управления крупными и сложными проектами разработки программного обеспечения. Вместо того чтобы создавать множество процессов управления для отслеживания прогресса и координации различных команд, вы сможете использовать рабочий программный код как функцию, принуждающую координироваться всю компанию. Требование ко всем программистам регулярно интегрировать свой код и с помощью автоматического тестирования гарантировать, что код всегда находится в рабочем состоянии, заставляет сотрудников координировать свои проекты разработки программного обеспечения без существенных издержек на управление.

Переход к практике работы с инфраструктурой как с программным кодом, которую привнесли Джез Хамбл и Дэвид Фарли и которая включает в себя контроль всех аспектов процесса разработки программного обеспечения с такой же строгостью, как и контроль непосредственно кода приложения, явился серьёзным прорывом. Этот подход требует, чтобы создание различных сред, развёртывание приложений и управление базами данных были автоматизированы программным кодом, который документирован и отслеживается в инструменте управления исходным кодом[1] точно так же, как и код самого приложения. Использование принципа «инфраструктура как код» создаёт единое понимание требуемых сред и процессов развёртывания в группах разработки, контроля качества и эксплуатации и обеспечивает согласованность на пути к продуктивной среде. И снова именно рабочий код помогает координировать различные участвующие в процессе группы сотрудников.

Переход к работе с инфраструктурой как с программным кодом увеличивает непосредственные коммуникации между разработкой и эксплуатацией, что является ключом к успеху всех видов культурных и структурных сдвигов, требуемых DevOps. Люди больше не входят в системы, чтобы внести неконтролируемые изменения. Вместо этого они работают вместе над общими скриптами для внесения изменений в инфраструктуру, и эти скрипты отслеживаются в инструменте управления программным кодом. Такой порядок требует от исполнителей, как минимум, документировать все изменения, чтобы каждый мог видеть, что же они делают, а в идеале он заставляет напрямую сообщать заинтересованным сторонам о вносимых изменениях, чтобы гарантировать, что такие изменения будут работать на каждом этапе конвейера развёртывания, вплоть до среды эксплуатации. Необходимость использования общего программного кода и общих инструментов заставляет людей сотрудничать. Не стоит недооценивать влияние такого сотрудничества на эффективность. Поскольку команды скоординированы необходимостью постоянного, ежедневного поддержания программного кода в рабочем состоянии, для решения возможных проблем создание специализированных процессов управления больше не требуется. Программное обеспечение, как известно, довольно трудно отслеживать с помощью таких процессов. Простое

[1] SCM, Source Code Management tool.

получение отчёта о состоянии проекта вызывает затруднения и требует больших накладных расходов. Более эффективно, если команды способны решить проблемы в режиме реального времени. Кроме того, гораздо легче отслеживать прогресс в разработке, используя конвейер развёртывания вместо написания множества различных отчётов для менеджеров, ведь теперь каждый может отслеживать состояние программного кода при его движении по конвейеру.

Применение строгого конвейера развёртывания, принципа «инфраструктура как код» и автоматизированного тестирования существенно отличается от подхода ITIL к управлению конфигурациями. Процессы ITIL были разработаны для обеспечения предсказуемости и стабильности, в то время как DevOps направлен на повышение скорости при сохранении надёжности. Самые большие различия касаются процессов управления конфигурациями и изменениями. Подход ITIL имеет очень строгие, исполняемые вручную процессы для всех изменений, которые происходят в продуктивной среде. Такие изменения обычно документируются вручную и утверждаются в специализированном инструменте управления изменениями через заявки или запросы. Утверждённые изменения затем вручную выполняются в среде эксплуатации. Такой подход помогает улучшить стабильность и целостность, но замедляет поток, так как требует множества передач работы и ручных операций. Принцип DevOps «инфраструктура как код» вместе с автоматическим тестированием в качестве контрольных точек, встроенных в конвейер развёртывания, позволяет лучше контролировать конфигурации, а также значительно повысить скорость. Это достигается через автоматизацию самого процесса, а также использование инструмента контроля программного кода. К примеру, предлагаемое изменение программного кода документируется с помощью скрипта в инструменте контроля программного кода. Критерии приёмки данного изменения документируются автоматическими тестами, которые также находятся в инструменте контроля. Кроме того, вы точно знаете, какие изменения были действительно реализованы, поскольку это было сделано с помощью скрипта автоматизации, попадающего под контроль версий. Подход предполагает размещение всего необходимого для управления изменениями в едином инструменте с необходимой автоматизацией, поэтому отслеживание становится более простым и быстрым. Также повышается строгость при утверждении изменений, поскольку теперь все критерии приёмки должны быть сформулированы в виде автоматических тестов, а не просто изложены в произвольном виде – и так для каждого изменения.

Дополнительные и значительные преимущества возникают в области аудита и соблюдения нормативных требований. Раньше аудиторская команда была вынуждена вручную отслеживать изменения, согласования и внедрение в различных инструментах; теперь же всё автоматизировано и отслеживается в едином инструменте. Это значительно упрощает обеспечение соответствия требованиям и нормам (compliance), потому что компьютеры намного лучше людей обеспечивают правильность исполнения процессов. Дополнительное облегчение команда аудита испытывает от того, что все изменения документируются в инструменте контроля программного кода, специально для этого и разработанного.

Описанные преобразования значительно повышают эффективность крупных организаций, поскольку они улучшают цепочку создания ценности при сохранении стабильности. Самое главное, однако, заключается в том, что создание и оп-

тимизация конвейера развёртывания подразумевают поиск и устранение потерь, которые существовали в вашей организации в течение многих лет. Улучшая поток, вы столкнётесь со множеством точек неэффективности, возникающей при координации работы между командами. Продуктивность работы отдельных сотрудников будет повышена за счёт лучшего качества и быстрой обратной связи, когда они пишут код, но самые большие выгоды будут получены от оптимизации координации работы внутри команд, между командами и между подразделениями. Это потребует внедрения специализированных технических средств, но, безусловно, самой большой трудностью является привлечение сотрудников на свою сторону и изменение их подхода к ежедневному выполнению своих обязанностей. Изменения будут значительными, но и выгоды будут существенными.

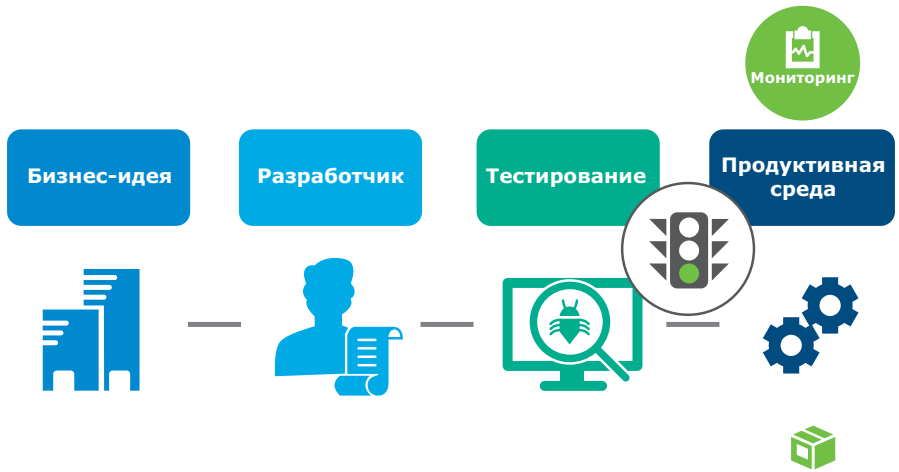
Резюме

Поскольку программное обеспечение становится основой конкуренции, применяемые нами подходы к управлению разработкой программного обеспечения напрямую влияют на скорость нашей реакции на инициативы бизнеса. Здесь нам может очень пригодиться DevOps. Суть заключается в повышении скорости при сохранении всех аспектов качества. По мере того как компании вступают на путь DevOps, они осознают всё многообразие идей относительно DevOps и применения новых подходов. Как показано в этой книге, большинство крупных компаний не имеет хорошей основы, чтобы применить идеи DevOps в своей организации. Это затрудняет взаимодействие сотрудников при внедрении изменений, которые могут улучшить систему в целом. Люди, работающие в крупных организациях, должны единообразно понимать цели, которых они хотят достичь при внедрении DevOps, и должны найти способы определения приоритетов улучшений. Иначе применение DevOps не принесёт ожидаемых результатов. Как будет показано в этой книге, документирование, автоматизация и оптимизация конвейера развёртывания в крупных компаниях и ИТ-подразделениях, разрабатывающих программное обеспечение, повышают эффективность и рациональность, а также предлагают очень хороший подход к трансформации управления крупными и сложными проектами разработки программного обеспечения.

Глава 2:

Базовый конвейер развёртывания

Конвейер развёртывания в крупных организациях может быть достаточно сложной для понимания и совершенствования системой. Поэтому имеет смысл начать с самого базового представления о том, что же такое конвейер развёртывания, чтобы разделить большую задачу на несколько простых и затем показать, как она масштабируется и становится всё более запутанной, по мере реализации её в крупных организациях со сложной структурой. Самая базовая конструкция конвейера – это протекание бизнес-идеи от разработки одним программистом через тестовую среду в эксплуатацию. Она показывает, как идея движется через ИТ-структуры компании, что является первым шагом к определению узких мест и потерь в системе. У некоторых людей может возникнуть соблазн начать конвейер развёртывания с разработчика, но я склоняюсь к тому, что он берёт своё начало с бизнес-идеи, поскольку мы не должны упускать из виду весь объём предъявляемых требований, неэффективность водопадного планирования и процедуру ежегодного бюджетирования, которые применяются в большинстве организаций.



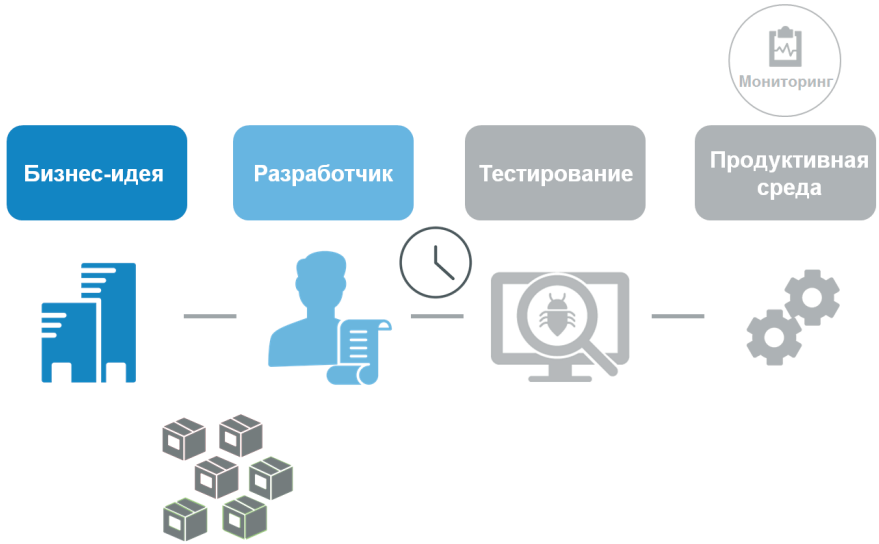
Первой ступенью конвейера является передача бизнес-идеи разработчикам, чтобы они могли реализовать новую функциональность. Затем, как только такая функциональность будет готова, разработчик должен протестировать её, чтобы убедиться, что она работает должным образом, что новый код не повредил уже работающей функциональности, не создал никаких дыр в безопасности и не повлиял на производительность. Это требует соответствующей среды. Затем код необходимо развернуть в тестовой среде и протестировать. Как только тестирование покажет, что новый код работает должным образом, он может быть развёрнут в продуктивной среде. Заключительным шагом является мониторинг приложения в среде эксплуатации с целью убедиться, что всё соответствует ожиданиям. В этой главе мы рассмотрим каждый шаг в данном процессе и возможные проблемы с эффективностью. Затем, в третьей главе, мы рассмотрим практики DevOps, которые призваны устранить встречающиеся недостатки.

Требования

Первый шаг в конвейере развёртывания – это переход от бизнес-идеи к началу разработки новой функциональности. Обычно это подразумевает также формирование списка требований и до некоторой степени – планирование дальнейшего развития. Первая проблема, с которой сталкиваются крупные организации при попытке получить выгоду от конвейера развёртывания, связана с тем, что они обычно склонны использовать водопадное планирование. Они делают это, поскольку уже используют водопадное планирование во всех других частях своего бизнеса и просто применяют те же принципы к ПО. Разработка программного обеспечения, однако, имеет три ключевых отличия. Во-первых, гораздо сложнее всё спланировать точно, поскольку всё, что вы просите сделать ваши команды, их просят сделать впервые. Во-вторых, если ПО правильно разработано с использованием строгого конвейера, его относительно быстро и недорого изменить. В-третьих, в нашей отрасли мы так плохо можем предугадать пользовательский опыт, что более 50% всего разработанного ПО попросту никогда не используется и не соответствует ожиданиям бизнеса. Благодаря этим отличительным особенностям программного обеспечения, если вы используете водопадное планирование, вы в конечном итоге блокируете свой самый гибкий и ценный ресурс, чтобы создавать функциональность, которая никогда не будет использоваться или не будет полезна бизнесу. Вы также затратите уйму сил на планирование, вместо того чтобы создавать реальную ценность вашему бизнесу.

Организации, которые используют водопадное планирование, также склонны выдвигать разработчикам множество требований. Эти требования имеют тенденцию замедлять поток получения ценности, создавать потери и вредить эффективности процесса. Как наглядно продемонстрировали в бережливом производстве, любые избыточные требования в системе, как правило, ведут к издержкам из-за переделок и суеты. Если организация инвестирует в постановку задач задолго до того, как они понадобятся, когда подключится разработчик, сформулированные требования зачастую приходится обновлять, чтобы ответить на вопросы, которые у разработчика могут появиться и/или чтобы соответствовать изменениям рынка. Это влечёт за собой дополнительные расходы и переделки в системе.

Другая проблема, связанная с наличием избыточных запросов к разработчику, заключается в том, что по мере изменения рынка приоритеты также должны меняться. Это приводит к тому, что организации должны либо постоянно заново формулировать запросы, либо, что ещё хуже, придерживаться намеченного плана и разрабатывать функциональность, которая с меньшей вероятностью отвечает текущим потребностям рынка. Организации, позволяющие процессу планирования заключить их в строгие оговорённые рамки, создают издержки, вызванные получением меньшей, по сравнению с ожидаемой, ценностью. А если организация пересматривает приоритеты большого числа требований, с высокой вероятностью утрачивают важность требования, на формулирование которых ранее были затрачены значительные ресурсы. В любом случае, избыточные требования приводят к дополнительным расходам.



Тестовая среда

Следующим шагом является создание среды, в которой новая функциональность может быть развёрнута и протестирована. Работа по созданию этой среды обычно относится к области компетенции специалистов по эксплуатации. В небольших организациях, использующих облака, это может быть очень просто и легко. В крупных компаниях, использующих внутренние центры обработки данных, процесс может быть очень сложным, требующим длительных закупок и согласований с вовлечением различных подразделений организации.

Создание среды может начинаться с длительных циклов закупок и крупных операционных проектов только для того, чтобы скоординировать работу команд, отвечающих за разные направления (серверы, хранилища, сети и прочее) в службе эксплуатации. Часто это является одной из самых ярких болевых точек, которые заставляют компании обратиться к DevOps.

Есть одна компания, которая начала внедрение DevOps с попытки понять, сколько времени потребуется, чтобы развернуть простое приложение *«Hello World!»* в среде, использующей их стандартные процессы. Они сделали это, чтобы понять, где находятся самые большие ограничения в их организации. Эксперимент был завершён через 250 дней, хотя *«Hello World!»* так и не поднялся и не взлетел, поскольку они почувствовали, что нашли самые большие ограничения. Затем они провели тот же самый эксперимент в Amazon Web Services и показали, что задача может быть решена за два часа. Этот эксперимент обеспечил хорошее понимание существующих проблем в организации и дал представление о том, что возможно, а что нет.

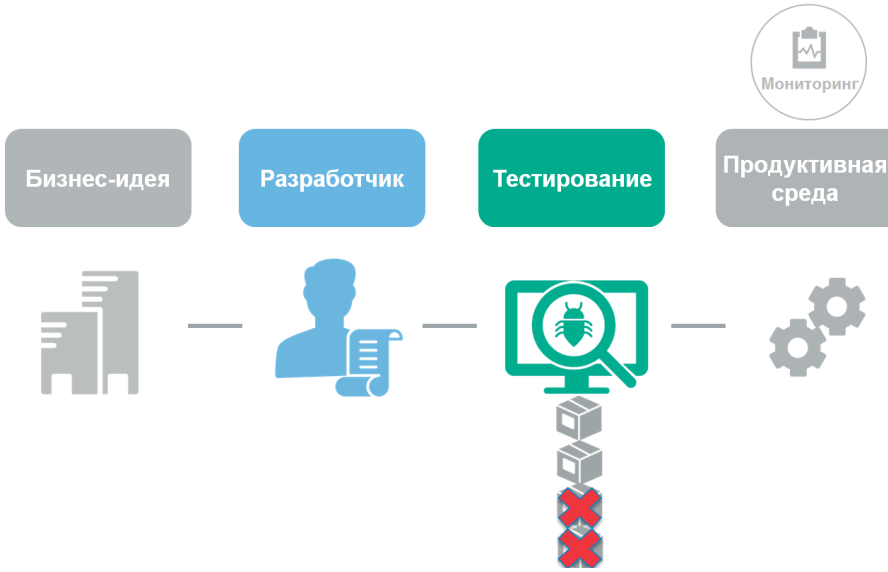
Тестирование и устранение неисправностей

Когда необходимая среда создана, следующий шаг – развёртывание кода с новой функциональностью в тестовой среде, чтобы убедиться, что код работает, как ожидалось, и не нанесёт урона существующим функциям. Этот шаг также должен гарантировать отсутствие проблем с безопасностью и производительностью, которые может создать новый код. Как правило, три проблемы мешают среднестатистическим организациям на этом этапе в их конвейере: повторяемость результатов тестирования, необходимость во времени для проведения тестов и необходимость во времени для устранения выявленных проблем.

Плохая повторяемость результатов является источником снижения эффективности большинства компаний. Они тратят время и энергию на отладку и пытаются найти проблемы с кодом, которые в конечном итоге являются проблемами, связанными со средой, развёртыванием кода или даже самим процессом тестирования. Это затрудняет понимание того, когда код может быть готов к вводу в эксплуатацию и требует дополнительных усилий. Крупные, сложные, с запутанной структурой организации часто тратят гораздо больше времени на настройку и отладку сред, чем на само написание кода для новых функций.

Тестирование обычно выполняется посредством дорогостоящих и трудоёмких ручных тестов, которые тяжело повторяемы. Вот почему важно автоматизировать тестирование. Время, затрачиваемое на полный цикл ручного тестирования, задерживает получение обратной связи разработчиком, что приводит к замедлению циклов повторной обработки и уменьшает поток в конвейере развёртывания. Время, которое тратится на эти ручные циклы тестирования и затраты на них, также заставляет организации объединять несколько новых функций вместе в крупные релизы, которые замедляют получение ценности для бизнеса и делают процесс вывода более сложным и неэффективным.

Следующей загвоздкой на данном этапе являются время и усилия, затрачиваемые на устранение всех недочётов кода, выявленных в тестовой среде, и получение приложений того уровня, который необходим для вывода в эксплуатацию. Для начала наибольшие ограничения накладывает время, затрачиваемое на выполнение всех тестов. Когда на это отводятся недели, разработчики обычно могут исправлять ошибки с той же скоростью, с какой тестировщики их находят. Но всё меняется, когда организация переходит к автоматизации, и время тестирования начинает измеряться часами. В этот момент узким местом становятся способности разработчиков быстро исправлять все ошибки и отправлять в среду эксплуатации код надлежащего качества.



Как только организации добиваются хороших результатов в создании сред или просто добавляют функции в приложение, которое уже настроено в среде, достижение надлежащего уровня качества становится одной из самых больших проблем, мешающих более частой публикации релизов. Я работал с компаниями, у которых есть команда разработчиков, практикующих крупные внутренние встречи, чтобы тестировать, отлаживать и готовить к выпуску программное обеспечение. Они встречаются каждый день, чтобы посмотреть, как идёт тестирование и когда они будут готовы к выпуску. Они отслеживают все ошибки и исправляют их, чтобы убедиться, что текущая сборка достигла нужного уровня качества. Зачастую вы можете видеть, что эти команды работают поздно вечером в пятницу, чтобы подготовить сборку к удалённому тестированию в выходные. Затем субботним утром обнаруживается, что команды тестировали неправильный код, или он был плохо развёрнут, или среда каким-то образом была сконфигурирована неверно. Такой процесс может привести к большому объёму работы в системе и настолько болезнен, что многие организации предпочитают проводить очень большие, но менее частые релизы, чтобы избежать проблем.

Вывод в среду эксплуатации

Следующий шаг, после того как весь код готов, – перевод его в среду эксплуатации для финального тестирования и приёмки в эксплуатацию заказчиком. Развёртывание – задача отдела эксплуатации, что важно, поскольку эксплуатация не всегда играет ведущую роль в DevOps-трансформации. Но в случае, когда вы используете конвейер развёртывания для иллюстрации того, как всё работает, становится ясно, что специалисты эксплуатации необходимы для трансформации и должны предпринять определённые действия для повышения эффективности процесса. Именно на этом этапе организации часто видят проблемы с при-

ложением впервые во время релиза. Нередко не ясно, связаны ли эти проблемы с кодом, развёртыванием, средой, тестированием или чем-то другим. Поэтому для развёртывания больших комплексных систем требуются крупные внутренние совещания для поддержки релизов. Кроме этого, сами по себе процессы развёртывания могут съедать много времени и ресурсов для ручных изменений. Затраты времени, усилий и количество проблем, связанных с этим процессом, часто подталкивают организации к пакетированию больших объёмов изменений в менее частые релизы.

Мониторинг и эксплуатация

Мониторинг – ещё одна задача специалистов эксплуатации, поскольку именно они владеют инструментами, которые используются в продуктивной среде. Часто мониторинг обеспечивается только там, что создаёт проблемы, поскольку, когда код уже развёрнут у заказчиков, разработчики могут не увидеть потенциальные проблемы прежде, чем их заметит сам заказчик. Если служба эксплуатации сотрудничает с разработчиками, чтобы обеспечить мониторинг «вне конвейера», потенциальные проблемы будут выявляться до того, как смогут повлиять на заказчика.

Когда код, наконец-то, выпускается и доступен заказчику, и в достаточной мере контролируется, чтобы гарантировать, что он работает должным образом, в идеале не должно возникать никаких новых проблем на этапе эксплуатации, если все тесты производительности и безопасности пройдены с хорошим результатом. В действительности это не всегда так. Например, я участвовал в одном крупном релизе, где мы провели обширное тестирование и тщательно контролировали процесс выпуска. Однако сразу после развёртывания в среде эксплуатации возник сбой, вызванный проблемой, которую мы никогда раньше не замечали. Каждый раз, когда мы направляли трафик заказчика через новый код, заканчивалась память, и всё падало. После нескольких попыток и сопутствующего сбора данных мы провели несколько часов, чтобы откатиться обратно к старой версии приложения. Мы знали, где ошибка, но даже когда мы пытались запустить процесс отладки, мы не могли воспроизвести её в наших тестовых средах. Через некоторое время мы пришли к выводу, что не сможем во всём разобраться, пока не вернём новый код в среду эксплуатации и не настроим мониторинг, чтобы найти проблему. Мы снова развернули код, и мониторинг показал нам, что у нас заканчивается память и всё рушится. На этот раз разработчики собрали достаточно данных, чтобы локализовать проблему. Оказалось, что один из разработчиков исправлял ошибку, которая некорректно обрабатывала длину текстовой строки. Команда, которую использовал разработчик, отлично работала во всех наших тестах, но в реальной эксплуатации мы поняли, что программа Internet Explorer 8, локализованная на испанском языке, имела дефект, который превратил эту команду в плавающую точку вместо целого числа, вызывая переполнение стека. Это был настолько уникальный случай, что мы даже не рассматривали его для тестирования. Кроме того, даже если бы мы его предусмотрели, запуск всех наших тестов в разных браузерах с разными локализациями был бы очень дорогостоящим. Такие проблемы напоминают нам, что конвейер развёртывания не завершён до тех пор, пока новый код не контролируется в продуктивной среде и не ведёт себя так, как ожидалось.

Конец ознакомительного фрагмента.
Приобрести книгу можно
в интернет-магазине
«Электронный универс»
e-Univers.ru