«Неважно, начинаете ли вы свой путь в DevOps или уже неплохо его освоили и хотите узнать, как применить его принципы в больших масштабах, – в процессе чтения книги вы несомненно оцените DevOps и его преимущества».

Д-р Баскар Гош

Во многих организациях формируется привычка к шаблонным методам работы, и сотрудники пасуют перед необходимостью совершенствования устаревшей ИТ-инфраструктуры. Мирко Херинг, специалист по управлению изменениями в области ИТ, предлагает отнестись к процессу обновления без предвзятости.

В своей книге Херинг рассказывает, как создать в компании правильную экосистему, организовать по-настоящему эффективную работу каждого сотрудника и правильно применять нужные технологии, которые поспособствуют развитию бизнеса.

Однако наличие правильных методов и инструментов не принесет успеха, если корпоративная культура останется прежней. Автор книги предлагает читателям практические упражнения, основанные на принципах Agile, Lean и DevOps, и описывает примеры из своей личной практики, которые помогут любой организации – малой или крупной, старой или новой – подготовиться к предстоящим преобразованиям, отказаться от консервативного стиля мышления и наладить ИТ-процессы таким образом, чтобы извлечь из них максимальную пользу.

Интернет-магазин: www.dmkpress.com

Оптовая продажа: КТК «Галактика»
e-mail: books@alians-kniga.ru

ДМК ПРЕСС издательство
www.дмк.рф

ISBN 978-5-97060-836-4

DevOps *для* современного предприятия

Мирко Херинг

DevOps для современного предприятия

Neoflex

ДМК ПРЕСС издательство

Мирко Херинг

DevOps
для современного предприятия

DevOps for the Modern Enterprise

Winning Practices to Transform Legacy IT Organizations

Mirco Hering

Foreword by
Dr. Bhaskar Ghosh

IT REVOLUTION
Portland, Oregon

DevOps для современного предприятия

Действенные практики для трансформации традиционных ИТ-организаций

Мирко Херинг

Предисловие
д-ра Баскара Гоша

ДМК ПРЕСС
издательство
Москва, 2020

Neoflex

УДК 004.45
ББК 65.290
Х39

Херинг М.
Х39 DevOps для современного предприятия / пер. с анг. М. А. Райтмана. – М.: ДМК Пресс, 2020. – 232 с.: ил.

ISBN 978-5-97060-836-4

DevOps – методика автоматизации рабочих процессов, существенно облегчающая задачи организации и способствующая действенным преобразованиям. Мирко Херинг, менеджер со стажем, рассказывает о том, как избежать распространенных ошибок на пути внедрения инноваций и добиться успеха в долгосрочной перспективе.

В первой части книги обсуждается экосистема предприятия, формирующая благоприятные условия для преобразований. Вторая часть посвящена работе с людьми, управлению ИТ-командой, внедряющей изменения, и выстраиванию рабочих процессов. В третьей части описываются технологические и архитектурные аспекты применения DevOps. В конце каждой главы приводятся практические упражнения.

Издание предназначено для менеджеров и ИТ-специалистов, занимающихся DevOps-проектами в организациях разного уровня.

УДК 004.45
ББК 65.290

ISBN 978-1-94278-819-5 (анг.)
ISBN 978-5-97060-836-4 (рус.)

Содержание

Впервые о DevOps заговорили в связи с переходом в эру цифровой экономики, когда скорость выпуска на рынок продуктов стала одним из ключевых конкурентных преимуществ. Технологиям, обеспечивающим стремительное развитие бизнеса, пришлось бежать со всех ног, чтобы только оставаться на месте, а для достижения дополнительных результатов – как минимум в два раза быстрее. Компаниям понадобились инструменты для быстрого и непрерывного улучшения качества существующих процессов разработки продуктов и их максимальной автоматизации, потому что хороший продукт стал равен хорошей ИТ.

Свой путь погружения в DevOps я начала несколько лет назад, когда возглавила отдел тестирования системы подготовки регулярной банковской отчетности Neoflex Reporting, которая отличалась большим количеством параллельных веток разработки и обилием ручных процессов. В ее разработку к этому моменту уже были вложены десятки тысяч человеко-часов.

Засучив рукава наша команда взялась за точечную автоматизацию этапов жизненного цикла продукта. В целом мы достигли неплохих результатов, но добиться слаженной и синхронной работы от всех участников процесса оказалось по-настоящему трудной задачей. Периодически возникающие «тут подкрутить», «там вручную запустить», «а это не на моей стороне», «я был на обеде», «исторически сложилось» тормозили ожидаемое от автоматизации ускорение.

Осознать, что же делать дальше, нам помогла книга, которую вы сейчас держите в руках. Мы прочитали ее всей командой и здорово переработали текущие процессы взаимодействия в парадигме слаженности, простоты и удобства. А процессы сборки, развертывания инфраструктуры, установки, тестирования и выдачи поставки объединили в непрерывный производственный конвейер.

«DevOps для современного предприятия» – книга об эффективной ИТ настоящего. Захватывающий и понятный путеводитель, способный обобщить, разложить по нужным полочкам существующий опыт и обогатить его ценными идеями.

В книге описаны основные шаги и принципы построения производственного взаимодействия, автоматизации процессов и развития культуры разработки ПО. Теория щедро сдобрена историями реальных людей и компаний, прошедших непростой, но интересный путь к DevOps.

Неоспоримая ценность этой книги в том, что она помогает вырваться из рутины бытия и взглянуть на текущие процессы совершенно другими глазами. Приходит осознание того, что на точечных «костылях» автоматизации далеко не уйти, появляется понимание того, как выглядит путь роста и развития, который подходит именно вашей компании, проекту, продукту.

Желаю вам приятного чтения, и пусть эта книга станет для вас источником неиссякаемого вдохновения!

Лина Чуднова,
руководитель практики DevOps компании «Неофлекс»

О компании «Неофлекс»

«Неофлекс» создает ИТ-платформы для цифровой трансформации бизнеса, помогая заказчикам получать устойчивые конкурентные преимущества в цифровую эпоху. Мы фокусируемся на заказной разработке программного обеспечения, используя передовые технологии и подходы.

Наш отраслевой опыт и технологическая экспертиза, усиленная собственными акселераторами разработки, позволяют решать бизнес-задачи любого уровня сложности. Среди наших заказчиков более половины российских банков, входящих в топ-100, а также компании из 18 стран Европы, Азии и Африки.

Телефон: +7 (495) 984 25 13
Сайт: www.neoflex.ru

Современный бизнес ожидает от ИТ все больше. Требования к производству программного обеспечения становятся жестче, планка качества – выше, времени и ресурсов – меньше. Бизнес хочет все сразу: Time2Market, снижение сбоев и отказов, и притом дешевле, а еще чтобы ИТ «прочитали мысли» и сделали правильно правильные вещи. Нужна магия, а время шаманов и энтузиастов ушло.

Необходим четкий процесс, который дает результат – когда это нужно заказчику или нужно быстрее. Все чаще информационные технологии не просто поддерживают бизнес, но становятся основным драйвером развития и трансформации.

На мой взгляд, DevOps не является революцией или универсальным лекарством. Это точно не магия, это лучше – качественный скачок, перевернувший принципы построения процессов производства ПО. Это то, что позволяет ИТ делать невозможное и при этом не ломаться от чрезмерного стресса и внутренних противоречий. Это вынужденная автоматизация рутинных операций. Это изменение требований к ИТ-специалистам, трансформация рынка труда ИТ и появление новой профессии «инженер DevOps».

Методики DevOps связывают ИТ в единый процесс, позволяя сделать его не только измеряемым и контролируемым, но и понятным и динамичным. Этот подход – если хотите, новая философия – связывает специалистов разных ИТ-областей в единую эффективную команду. Вы можете сосредоточиться на управлении, исходя из принципа достижения глобального оптимума, а не частных оптимумов на разных участках ИТ-производства. Узкие места процессов выходят из тени, а оптимизация становится неизбежной.

В цифровую эпоху скорость и возможность динамического изменения имеют первостепенное значение. Время – это не просто деньги. Зачастую это часть стратегии, основа конкурентной борьбы, а динамика изменений – это конкурентное преимущество, формирующее уникальность. В этой книге вы найдете советы и примеры, что позволит погрузиться в мир DevOps. Книга захватывает и вовлекает, так что вы сами не заметите, как DevOps станет частью вашей жизни и войдет в культуру производства.

Искренне желаю вам увлекательного чтения. Узнавайте, практикуйте и пробуйте подходы DevOps уже завтра. Поверьте, это полезно, и это пригодится!

Владимир Туровцев,
управляющий партнер, директор по развитию бизнеса Logrocon,
ведущий преподаватель программы MBA школы бизнеса МИРБИС,
победитель конкурса «Лидеры России» 2018–2019

О компании

Компания Logrocon – Microsoft Gold DevOps Partner. С 2012 года разрабатывает, тестирует и внедряет ПО. Организатор международной конференции DevOps Forum 2019, международных некоммерческих комьюнити-конференций DevOpsDays Moscow 2018 и 2017. Компания реализовала более 40 проектов по DevOps.

Телефон: +7 (495) 777-00-84
Сайт: https://logrocon.ru/

Соцсети:
https://www.facebook.com/logrocon/
https://www.facebook.com/OrangeOceanIT/

YouTube-канал: http://www.youtube.com/c/Логрокон

Сайт организованных в 2017–2019 гг. конференций: https://devopsforum.ru/
Сайт иммерсивной ИТ-площадки «Оранжевый океан»: https://orange-ocean.ru/

ОТ ИЗДАТЕЛЬСТВА

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в основном тексте или программном коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу dmkpress@gmail.com, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Об авторе

На протяжении десятков лет Мирко Херинг работал над ускорением доставки программных продуктов при помощи инновационных подходов (в современной терминологии – DevOps); десять лет назад он начал экспериментировать с Agile-методиками. Будучи руководителем по DevOps и Agile по Азиатско-Тихоокеанскому региону в компании Accenture, он оказывает поддержку крупным публичным и частным компаниям по всему миру, помогая им в поисках эффективных методов ИТ-доставки. Мирко ведет блог **NotAFactoryAnymore.com** и выступает на международных конференциях, делясь своими знаниями и опытом. Вы также можете подписаться на его страницу в Twitter: **@MircoHering**.

Благодарности

То, что любят говорить о воспитании детей, справедливо и для написания книги – такую работу в одиночку не провернуть. И наверняка я упустил кого-то из тех, кто достоин упоминания на этих страницах. Когда мы встретимся на следующей конференции, с меня причитается угощение!

Прежде всего хочу поблагодарить фантастическую команду, которая поддерживала меня на протяжении всего процесса редактуры: Тодда, Джина, Лею и Карен – без вас мои мысли никогда не уложились бы в связный текст. Это был нелегкий труд, но работать с вами было мне в радость!

Еще хочу сказать спасибо за отзывы коллегам, которые посвятили этому свое время: Эрику, Енгу, Эджею и Эмили. Вы помогли отшлифовать конечный результат без ущерба для смысла.

Особую благодарность я хотел бы высказать трем ребятам, без которых это издание не состоялось бы. Эрик, Тодд и Джин помогли мне преодолеть установку «я никогда не напишу стоящей книги» и сфокусироваться на мысли «у меня и правда есть что поведать миру». Без вашей поддержки я не решился бы приступить к делу.

Также я хочу выразить благодарность руководству компании Accenture – Баскару Гошу, Адаму Бердену и Питеру Ваккасу – за поддержку этого проекта и предоставление мне гибкого рабочего графика. В результате я мог отвести достаточно времени на свое «хобби».

Спасибо Гэри Груверу, который поделился со мной советами по поводу написания книги и помог найти оптимальную схему работы.

В дополнение хочу поблагодарить всех сотрудников Accenture – практикующих DevOps и всех прочих – за то, что они помогли мне выработать свой подход к трансформированию компаний. Хочу сказать спасибо клиентам, с которыми я работал и у которых я постоянно учусь чему-то новому. Некоторые из ваших идей нашли свое отражение в этой книге.

И последнее, но самое важное: огромное спасибо моей жене Аньяли, которая здорово поддерживала меня – даже тогда, когда ее внимание было поглощено очаровательным младенцем, появившимся на свет как раз в дни написания этой книги. Аньяли, ты просто чудо!

Предисловие д-ра Баскара Гоша

На протяжении многих лет работы в ИТ-сфере я наблюдал не один кризис, связанный с появлением новых технологий, бизнес-моделей, а также со сменой глобальных экономических циклов. На волне подобных перемен DevOps приобретает особую значимость. Он замечателен не только своими принципами, но и изменениями, к которым ведет.

Мне нравится говорить, что DevOps можно использовать без Agile, но Agile нельзя использовать без DevOps. Это один из многих процессов трансформации, которые были запущены на заре DevOps. Помогая разработчикам наращивать эффективность и, соответственно, добиваться лучших результатов, DevOps способствует творческому самовыражению, что, в свою очередь, приводит к постоянному улучшению процессов поставки систем.

И хотя я люблю вспоминать те времена, что я провел, управляя процессами инфраструктур в крупных организациях, дело существенно осложнял тот факт, что мне приходилось разделять обязанности между разработкой и управлением процессами. Конечно, это была эпоха монолитных систем, использующих методологию водопада в больших масштабах и периодическую поставку программных пакетов. Для такой среды разработки разделение обязанностей представляло практичную и эффективную модель, которая соответствовала темпам изменений в системе.

Однако в цифровую эпоху скорость приобретает первостепенное значение. Разделение обязанностей сегодня не содействует все более ускоряющейся смене подходов, которые на данный момент необходимы бизнесу.

В этой книге Мирко раскрывает больше, чем просто технику DevOps, – он делится своей страстью к улучшению процессов проектирования программных

продуктов. Используя понятные аналогии и рекомендации, Мирко показывает, как помочь компании внедрить DevOps. Не важно, начинаете ли вы свой путь в DevOps или уже неплохо его освоили и хотите узнать, как применить его принципы в больших масштабах, – в процессе чтения книги вы несомненно оцените DevOps и его преимущества.

Д-р Баскар Гош,
исполнительный директор Accenture Technology Services
Бангалор, Индия
Март 2018 года

Предисловие автора

Обучение – дело не обязательное, а добровольное.
Совершенствование – дело не обязательное, а добровольное.
Но чтобы выжить, мы должны учиться.

Уильям Эдвардс Демминг

Одним из самых благодарных занятий в моей карьере был поиск наиболее эффективных способов создания значимых проектов и передача этой информации как можно большему кругу круг людей. Когда мы работаем неэффективно, то тратим время на ненужные вещи, на повторяющиеся и скучные задания, что не приносит никакого удовлетворения. Я не вынашиваю цели изменить мир, побуждая людей развлекаться на работе, но, мне кажется, труд должен быть человеку в радость. Когда сотрудники занимаются любимым делом, хороший результат не заставит себя ждать.

С того времени, как я пришел работать консультантом в Accenture – а было это более десяти лет назад, – я успел поработать с десятками команд, увеличивая эффективность и темп их работы, а следовательно, и качество поставки продукции. Но и прежде, чем стать консультантом, я думал о том, как сделать процессы в ИТ более эффективными.

На заре моей трудовой деятельности, в конце 1990-х, в ИТ только начинали привлекать к работе сторонних сотрудников. Первые несколько лет я набирался опыта в исследовательских лабораториях IBM, работая в области *телематики* и над инструментами для разработчиков (например, разрабатывая языки для особых CPU и предоставляя соответствующие *компиляторы* и *IDE-расширения*). Когда я начинал, разработка пакетного программного обеспечения была развивающейся отраслью, но основной объем работы осуществлялся

для частной разработки и штатно. Единственным способом увеличения продуктивности было усовершенствование автоматизации, и во всех моих ранних проектах мы выстраивали креативные решения при помощи различных shell-скриптов, Perl-скриптов и других оригинальных инструментов, которые облегчают жизнь разработчикам и администраторам. Я был очень доволен, когда выстраивал все эти решения по автоматизации и видел, как всем участникам проекта становится легче и веселее работать.

Был любопытный случай. В течение пяти лет я был занят в двух больших проектах и работал над созданием таких инструментов для разработчиков, которые позволяли успешно поставлять продукты. Когда я закончил работу на этих проектах, то начал оценивать разные организации, и мне показалось, что того уровня автоматизации на данный момент недостаточно. Все-таки я всю свою профессиональную жизнь решал задачи по автоматизации для команд поставки. Когда я говорил об этом с коллегами, мне становилось ясно, что пакетное программное обеспечение и возможности нештатной поставки открывают путь к увеличению продуктивности и снижению стоимости, чем многие организации пользуются, вместо того чтобы инвестировать в хорошие практики разработки и инструменты[1]. Я потратил следующие несколько лет, осваивая ту нишу на рынке, которая, как считалось, помогает организациям реализовывать инструменты для поставки. Но, по правде, организации не были в восторге от предложения вкладываться в это.

Предложение увеличить объем нештатных поставок и сократить среднюю стоимость рабочего дня разработчика (известную также как *средняя суточная ставка*) казалось более привлекательным для организаций, чем нечто более сложное и трудноизмеримое – разработка хорошей платформы поставки, которая помогает всем повысить производительность труда в ИТ-сфере. В конце концов, как вы будете измерять производительность в ИТ? Я солидарен с Рэнди Шупом и Эдрианом Кокрофтом, которые на конференциях признавались, что на протяжении всей своей карьеры пытались выявить достойный показатель производительности, но не смогли найти ничего стоящего. Я поднимал эту тему в блоге, рассказав о том, что производительность в ИТ очень сложно измерять; вместо этого стоит оценивать время выполнения циклов, неоправданные расходы и поставленную функциональность. Важно брать во внимание некоторые значимые метрики, иначе вы не сможете наблюдать за вашим развитием, поэтому получается, что в общем случае производительность в ИТ – достаточно расплывчатое понятие, и нам необходимо искать другие показатели, которые помогут нам судить о том, насколько наша работа эффективна.

В дальнейшем я провел несколько лет, работая над тем, чтобы понять, откуда растут проблемы в ИТ и как их решить. Мне повезло, так как мои исследования совпали со временем распространения технологий и методологий,

[1] Подробнее об этом рассказывается в моей статье на DevOps.com: Why We Are Still Fighting with the Same Problems in DevOps as 15 Years Ago («Почему мы не можем справиться с проблемами в DevOps, существовавшими еще 15 лет назад»).

которые впоследствии заложили основы нового способа реализации поставки в ИТ: Agile, DevOps и *облако*, среди прочего, значительно упростили реализацию тех решений, которые я построил за всю свою карьеру. Ниша, в которой я работал, становилась все более привлекательной, – сегодня сложно найти организацию, которая не говорит об Agile и DevOps.

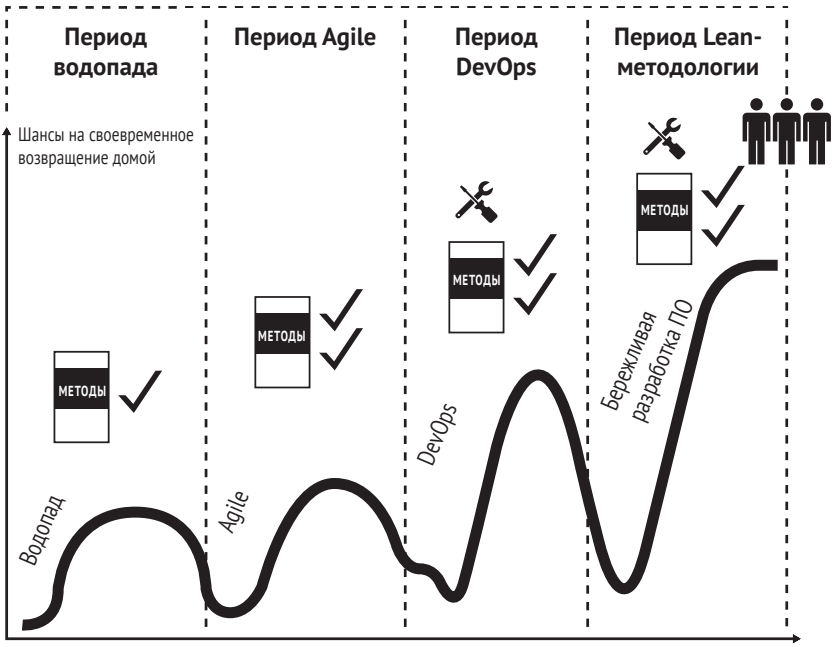
Но, оглядываясь назад и оценивая, где сегодня находится ИТ-индустрия, мы понимаем, что ИТ-поставка все еще не там, где она должна быть. Мы склонны считать, что *непрерывная поставка* является прекрасной практикой создания современных архитектур приложений, но если мы поищем организации, которые в совершенстве ей овладели, то обнаружим, что их совсем немного; к тому же их непросто найти. Многие организации выстраивают рабочие процессы, которые долгое время развивались с оглядкой на методы традиционного производства. В конце концов, эти практики были хорошо систематизированы во многих учебных планах MBA и накопили сотни лет практики. Но данные подходы и идеи уже не актуальны.

Я сам не овладел этим в совершенстве и все еще упорно учусь каждый день, но хочу, чтобы мои упражнения были доступны как можно большему кругу людей. Как вы видите на рис. 1, я разработчик до мозга костей: в первую очередь разрабатываю технические решения, вместо того чтобы думать о заинтересованных людях. Я на своих ошибках понял, что одно лишь применение правильных методов и инструментов не способно магическим образом преобразовать организацию. Сменить культуру – самая сложная задача, но также и самая значительная. Мне понадобилось совершить множество ошибок и промахов, чтобы узнать то, что я собираюсь раскрыть в этой книге, и понять, что необходимо вести преобразования только в готовой к этому культуре, дабы достичь устойчивого эффекта.

Последние несколько лет я разрабатывал практикум, который провожу для CIO и других ИТ-руководителей, чтобы показать их проблемы и помочь определить возможные пути решения. Самое замечательное в том, что ты встречаешься с умными и прогрессивными лидерами, – это возможность каждый раз узнавать нечто новое. Я уже проводил данный практикум в течение некоторого времени и невероятно благодарен за опыт и идеи, которые передали мне CIO: они помогают мне совершенствовать мой практикум. (Материал этой книги частично отражает и знания, полученные в ходе таких сессий.)

DevOps для современных компаний ссылается на различные испытания, с которыми сталкиваются организации во время их преобразования в современные ИТ-организации. А такая трансформация сейчас характерна для многих организаций: будь то производители автомобилей или банки, их бизнес зависит от ИТ. Мы видим, что технологии развиваются все быстрее и быстрее. Но в то же время у нас есть *устаревшие* приложения, оставшиеся с давних времен. А новые приложения, создаваемые сегодня, устареют через несколько лет. Я даже согласен с идеей о том, что устаревшим можно считать код, который был написан вчера! Программное обеспечение и технологии меняют ландшафт бизнеса, подсказывая людям новые способы взаимодействия и об-

мена информацией. Более того, технологии позволили нам преодолевать препятствия и расстояния. В итоге мир становится более сложным, он меняется быстрее, и из-за этого новые типы потребления разрушают установившиеся бизнес-практики. Многие организации сталкиваются с фундаментальной необходимостью модернизации ИТ-инфраструктуры. Наши старые *ментальные модели* и методы больше не работают. Нам нужно по-новому подходить к необходимости поиска новых решений. Но путь к современным решениям и улучшению технологии все еще кажется туманным и непростым. Очень немногие организации старого образца сегодня прошли через это преобразование.



Водопад	Agile	Практики DevOps	Культура DevOps
Методы и процессы		Инструменты DevOps	Люди
Отлаженные процессы		Усовершенствованные процессы	Применение принципов

Рис. 1. Развитие представлений Мирко о переменах в организации

На протяжении своей карьеры я успел поработать с рядом крупнейших технологических организаций в каждой из вертикалей этой индустрии. И какие бы вы ни слышали оправдания, будь они связаны с технологией, сложностью или культурой, поверьте мне, я видал ситуации и похуже! Тем не менее эти

организации были способны к радикальному преобразованию и улучшению своих результатов. Хочу поделиться с вами некоторыми из открытий и достижений. ИТ не должна быть областью, в которой большая часть времени тратится на решение одних и тех же проблем.

Материал этой книги основан на принципах Agile, DevOps и Lean. На самом деле большинство разговоров с клиентами я начинаю с вопроса «Что эти принципы значат для вас?», ведь все эти методологии такие неоднозначные. Я часто применяю схему, показанную на рис. 2, чтобы ясно очертить для слушателей, как я пользуюсь этими принципами.

Выполняя роль консультанта, я в то же время поставил себе цель как можно больше дистанцироваться. Как только мои клиенты начнут успешно применять принципы Lean, DevOps и Agile, а также выходить на более эффективную работу, я смогу пойти дальше и посвятить свое время реализации существующих решений. Я не могу представить себе лучшей цели в жизни, чем предоставлять заказчику максимальную свободу и вовремя отступать, сосредоточиваясь на одном-двух новых проектах.

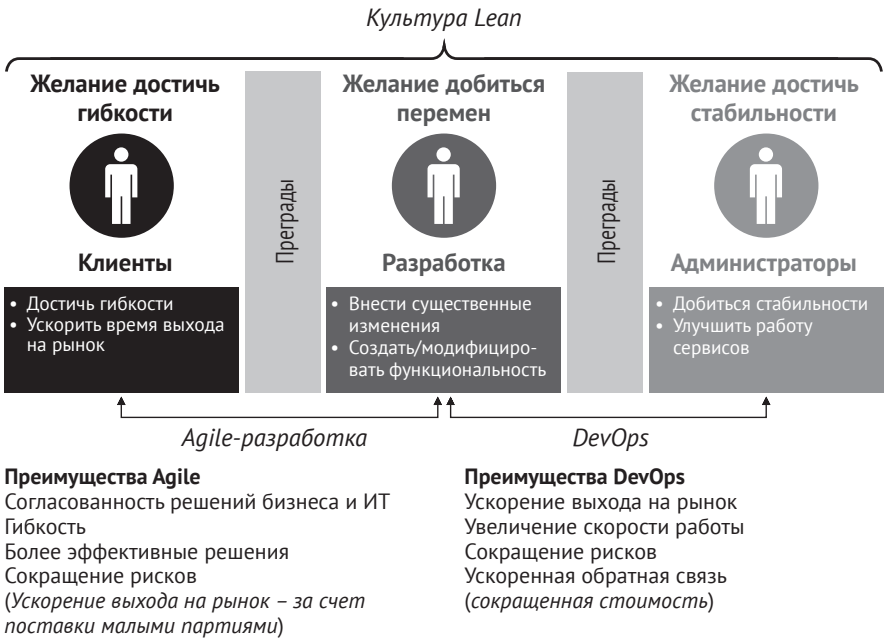


Рис. 2. Взаимосвязь Agile и DevOps:
как принципы Lean, Agile и DevOps связаны друг с другом

Но пока в сфере ИТ слишком многое все еще делается неправильно и людям приходится каждый день преодолевать проблемы, я буду принимать участие в совершенствовании мира ИТ. Я буду помогать организациям пережить преобразования при помощи этой книги, ведь моя команда не может быть вез-

десущей. У каждого из нас работы предостаточно. Честный ответ на вопрос о сложных проблемах состоит в том, что описанные в книге методы – только начало. Вы можете экспериментировать, добавлять что-то от себя и использовать мои советы частично. Каждое преобразование зависит от контекста, и не существует универсального рецепта успеха.

Надеюсь, что эта книга поможет вам с честью преодолеть неизбежные трудности. Я делюсь своим опытом и практическими упражнениями, и вы можете использовать их в вашей организации (будь она крупная или небольшая, старая или молодая), чтобы добиться улучшений. Я с нетерпением буду ждать встречи с вами на этом пути: на конференциях, где вы расскажете о своих успехах и провалах, на митапах или, возможно, на одной из моих консультационных встреч, где мы вместе будем искать решения ваших вопросов.

Я разработчик до глубины души и хочу создавать полезные приложения. Давайте сообща менять нашу индустрию, чтобы мы могли больше времени посвящать созидательной стороне ИТ.

Мирко Херинг

ВВЕДЕНИЕ

Как мы к этому пришли

> Основной опасностью кризисных времен является не кризис, а то, что вы будете действовать согласно вчерашней логике.
>
> *Питер Ф. Дракер. Managing in Turbulent Times («Менеджмент во время кризиса»)*

Прежде чем подобраться к самой сути, я хотел бы объяснить, почему важно менять подход к поставке ИТ-услуг. Мы явно не планировали ситуацию, которая сложилась на сегодняшний день, когда много организаций прикладывают массу усилий, чтобы поставлять продукты, приносящие пользу бизнесу. Услуги ИТ либо запаздывают, либо слишком дорого стоят, либо не способны обеспечить то качество, которого ожидают заинтересованные лица на стороне бизнеса, и это не потому, что мы сознательно принимаем ошибочные решения. Поразмыслив, вы поймете, что мы все примерно одинаково представляем себе, как выглядит хорошая работа, – примеры Netflix и Google встречаются в сотнях презентаций об ИТ. Почти каждая ИТ-организация говорит об Agile и гибкой поставке, автоматизации в жизненном цикле поставки программного обеспечения (англ. *Software Delivery Life Cycle*, сокр. SDLC) и применении современных паттернов архитектур, таких как *cloud-native приложение*, приложение двенадцати факторов и *микросервисы*.

Однако нам с трудом удастся отыскать организации, которые в совершенстве овладели этим новым методом работы в ИТ. Многие компании, которые могут похвастать положительным опытом, относительно молоды и изначально развивались как облачная или интернет-платформа (компании, которые со времен основания делали ставку на интернет как основную платформу). Мы даже придумали для них название: DevOps-единороги. Потому что встретить их не проще, чем единорога, и их уровня слишком тяжело достичь средней организации с устаревшей ИТ-архитектурой.

Можно подумать, что основным испытанием для такой организации выступает трансформация старой архитектуры. В определенной степени это верно, но я склоняюсь к тому, что основная проблема – в менталитете. Многие лидеры в технологической отрасли используют идеи, заимствованные из области традиционного производства, несмотря на то что ИТ по своему характеру от него отличается[1], – они применяют ментальную модель производства для творческого, по своей сути, процесса. В традиционном производстве мы следуем предсказуемым процессам для получения одного и того же результата (продукта), раз за разом. В ИТ, наоборот, мы не занимаемся одним и тем же проектом дважды. Чтобы следовать созидательной природе ИТ и добиться изменений менталитета в компаниях, нам нужно обратиться к следующим трем областям: к организационной экосистеме, в которой мы находимся, к людям, с которыми работаем, и к технологиям, на которых держится наш бизнес. Этот трехчастный подход и определил структуру книги. Во введении я объясню, почему прежний стиль мышления более не актуален. Далее вы найдете рекомендации, которые помогут вам преобразовать свою организацию и свойственную ей линию поведения.

Структура книги такова:

1. Часть А «Создание подходящей экосистемы» (главы 1–4): как вы сформируете организацию, которая должна быть способна обслуживать ИТ? Как вам необходимо работать с другими организациями, такими как поставщики программных продуктов и компании – интеграторы решений, для того чтобы соответствовать новому способу ИТ-поставки? Первая часть книги дает ответы на эти вопросы и адресуется CIO и другим лидерам в ИТ, тем, кто задает стратегию организации.
2. Часть Б «Люди и организация процессов» (главы 5–8): в организации ничего не происходит без участия людей. Мы так мало задумываемся об этом, что называем сотрудников «ресурсом»! В части Б этой книги мы сосредоточимся на людях и на том, как воодушевить их на ежедневное совершенствование: ведь в этом, помимо прочего, залог успеха организации.
3. Часть В «Технологические и архитектурные аспекты» (главы 9–12): я уверен в том, что недавние достижения в области технологий существенно расширяют способы работы и принуждают лидеров индустрии переосмысливать методы поставки ИТ-решений. Я расскажу о некоторых ключевых тенденциях и о том, как наилучшим образом ими воспользоваться. Технологии развиваются с невероятной скоростью, поэтому я не стану упоминать ничего конкретного – инструменты, поставщиков или методы. Чтобы помочь вам сориентироваться в этом пространстве, я все же предоставлю список ресурсов, который, надеюсь, не утратит актуальности с течением времени.

---

[1] Некоторые книги о DevOps, такие как *The Phoenix Project*, описывают современные модели производства, которые отличаются от этой традиционной модели.

Каждая глава основного блока содержит обсуждение определенной проблемы и предлагает возможные решения и подходы. Я старался делать главы покороче, чтобы проще можно было усвоить материал.

В конце каждой главы приводятся упражнения в стиле «сделай сам»: я предлагаю вам некоторые практические шаги, позволяющие применить знания, почерпнутые из соответствующей главы. Вашему вниманию представлены шаблоны, некоторые вопросы, которые вы можете задать себе или вашей организации, или пошаговое руководство. Это не заменит сотрудничества с опытным консультантом, но поможет совершить первые шаги на пути глобального преобразования.

Я также привожу ссылки на некоторые из моих статей там, где широко раскрываю некую тему, но желающим изучить ее глубже могли бы понадобиться дополнительные подробности. Можете полюбопытствовать и прочитать эти статьи.

Также в книге вы встретите фрагменты, резюмирующие вышесказанное и предлагающие некоторые общие рекомендации в отношении ваших следующих действий. Я уже говорил о том, что был вынужден изыскивать собственные методы, позволяющие развивать скорость работы в ИТ-индустрии. По сути, информация, приведенная в книге, должна подготовить вас к началу преобразования вашей компании в современную ИТ-организацию.

И наконец, в книгу включено приложение, где описываются подробные аналогии с заводами (фабриками), о которых говорится на протяжении всей книги. Если это совершенно новая для вас тема, вам, вероятно, стоит сначала прочесть приложение, перед тем как подойти к основному содержанию книги. Тем, кто хорошо знаком с такими аналогиями, приложение поможет понять их глубже и, возможно, сделать полезные для себя выводы. А теперь давайте немного поговорим о том, почему схемы, применяемые в традиционном производстве, помогают достичь быстроты и надежности ИТ-поставки, к чему мы и стремимся, используя DevOps.

Принципы традиционного производства не так просто применить в ИТ-поставке

В последние годы методы современного традиционного производства вдохновляли новаторов на появление таких течений, как Lean, системное мышление и теория ограничений. Производственная система компании Toyota на самом деле послужила одним из источников идей, которые переросли в Agile и DevOps. Тем не менее многие менеджеры занимаются управлением, все еще черпая вдохновение в старых подходах, не упуская из виду образ модели производства, сформировавшейся еще во времена Генри Форда. Она основана на идее о том, что мы выстраиваем процесс производства, чтобы максимально рационализировать процесс управления, в результате чего можно нанять менее квалифицированных работников, но добиться наиболее качественных

результатов. Однако в сфере ИТ, напротив, для достижения наилучших результатов необходимы сотрудничество и креативность. Мне кажется, что старые принципы производства привели ко множеству проблем, с которыми мы сегодня сталкиваемся в ИТ: например, строгие процессы с административным стилем управления, раздутые и сгруппированные по функциональным навыкам организационные структуры, а также частое делегирование обязанностей угнетают рабочий поток и повышают риск технологических ошибок и сбоев.

Я назвал свой блог «больше никаких заводов» (notafactoryanymore.com), чтобы таким образом призвать к переменам в мышлении, которые необходимы исполнительным ИТ-директорам для успешного преобразования организаций. Слово «завод» (или «фабрика», factory) отсылает к тому типу производства, который когда-то предложил Генри Форд: массовое производство осуществляется специалистами, работающими на сборочной линии и выполняющими узкоспециальные задачи, отчего при производстве продукта очень редко происходят какие-либо изменения, если вообще происходят.

Устаревшие методы производства были основаны на изобретении Фордом сборочных линий и на работе, написанной Фредериком Тейлором, о принципах научного менеджмента[1]. Запуск завода был дорогостоящим и длительным процессом. Сама работа на заводах оптимизировалась так, чтобы работники направляли все свои усилия на выполнение одной узкой задачи и могли научиться осуществлять ее невероятно эффективно. Для сокращения затрат производитель мог бы инвестировать в автоматизацию производства, приобретая новейшее оборудование; он мог бы сменить материал для изготовления продукта или переместить свой завод туда, где рабочему можно было бы платить меньше. Любые изменения в отношении самого продукта или процесса производства превращались в настоящее испытание, так как для этого приходилось приобретать новые машины, перенастраивать их, обучать всех работников новым процессам или менять схемы поставки материала для продукта.

Сам продукт практически не менялся (даже Генри Форд говорил о модели Т: «Вы можете получить “Форд-Т” любого цвета, при условии что этот цвет будет черным»), поэтому было достаточно просто сравнить результаты работы и стоимость продукта. Если вы измените процесс производства или начнете использовать другой материал для изготовления продукта, то можно с научной точки зрения оценить результаты (как раз эту концепцию предлагает работа Тейлора). Можно поставить под сомнение, что современное производство все еще позволяет вам измерять стоимость и продуктивность, так как все продукты на сегодняшний день производятся массово. Это тот тип производства, на котором основаны все экономические программы обучения и, соответственно, менеджмент. Я поступил в университет около пятнадцати лет назад, а недавно получил степень MBA, поэтому знаю не понаслышке, что обучение

[1] Фредерик Тейлор считается основателем научного менеджмента, который он реализовывал в стальной промышленности. Он опубликовал свою работу «Принципы научного менеджмента» в 1911 году.

ИТ-процессам все еще следует этой модели. Так или иначе, такую модель легко понимать и контролировать, что позволяет использовать научный подход к процессам.

В самом начале ИТ-отрасль имела много общего с производственным бизнесом. Давайте вернемся в 1990-е и зайдем в ИТ-отдел автомобильной компании. (Я предлагаю подобный сценарий, потому что в школьные годы работал в таком отделе по выходным и именно благодаря этому решил строить карьеру в ИТ.) Рабочие процессы в ИТ в целом организовывались аналогично другим производственным процессам и следовали тем же тенденциям. Создание новой ИТ-системы обходилось дорого и занимало много времени. Сама работа оптимизировалась таким образом, чтобы сотрудники могли стать хорошими специалистами по тестированию, Java-разработчиками, SAP-конфигураторами или инженерами по эксплуатации. Каждый из них мог сосредоточиться на одной специфичной задаче и выполнять ее невероятно эффективно в рамках процесса разработки по методологии водопада.

Для сокращения расходов ИТ-отдел мог предпринимать следующие шаги: можно было инвестировать в автоматизацию, приобретая улучшенные инструменты, сменить используемый тип программного обеспечения (например, перейти с частной системы на SAP, чтобы пользоваться уже готовой функциональностью, вместо того чтобы создавать ее самостоятельно), а позднее можно было бы перенести разработку туда, где сотрудникам можно меньше платить. В большинстве случаев внесение изменений в продукт сулило большие испытания, так как это подразумевало изменение условий контракта с поставщиками ПО или значительное преобразование существующей команды.

Сам продукт представлял собой нечто стандартное: *систему планирования ресурсов предприятия* (*ERP*) или *систему управления взаимоотношениями с клиентами* (*CRM*), использующую наилучшие практики, основанные на готовых решениях. И хотя выпускаемые продукты не были идентичными, общие усилия для их создания были сопоставимы (например, реализация SAP в одной автомобильной компании была сравнима с реализацией SAP в какой-либо другой компании).

Как вы видите, несмотря на то что ИТ – особая сфера, она все равно пересекается в управленческой части с другими отраслями, поскольку в ней используются практики, взятые из традиционного производства. И они работают уже довольно продолжительное время. Но за это время сфера ИТ уже сильно изменилась, а мы все еще не пытаемся управлять ей как-нибудь по-новому.

Применение в ИТ-области «заводских» методов порождает следующее представление о рабочем процессе: сотрудники ИТ-отдела сидят за перегородками и колдуют каждый над своей задачей; один человек преобразует требования в проект, передает его другому человеку, который пишет некоторый код, а затем передает его еще одному человеку, который его протестирует, – и при всем этом никто друг с другом даже не разговаривает. Все делается механически, как на конвейере. (Здесь вспоминается Чарли Чаплин в фильме «Новые времена».) Мир меняется, и на сцену выходит человек, готовый нести перемены:

сокращать усилия и расходы, затрачиваемые в ИТ на операции и подготовительные мероприятия.

Понятия операционных издержек и объема работы выступают основой для смены подхода

Операционные издержки (их еще называют издержками на подготовку) – это расходы, необходимые для преобразования механизма производства, чтобы начать выпускать другой продукт или сменить способ изготовления продукта. Размер операционных издержек определяет оптимальный объем работ по продукту, что позволяет добиться определенных экономических результатов, в то время как на издержки на хранение воздействовать довольно сложно. Возможное влияние операционных издержек на оптимальный объем работ показано на рис. 3[1].

На определение оптимального объема работ влияют издержки на хранение и операционные расходы (чем больше издержки на хранение, тем меньше будет оптимальный объем работ, а чем больше операционные расходы, тем оптимальный объем работ будет больше). В ИТ издержки на хранение являют собой комбинацию возрастающей стоимости исправления дефектов по мере продвижения по жизненному циклу разработки и упущенной выгоды от того, что готовая функциональность простаивает, будучи все еще не выпущенной в среду эксплуатации. Эти два фактора не сильно изменились с приходом практик DevOps; чего коснулись изменения, так это операционных издержек.

По мере того как традиционное производство менялось с течением времени, начала усиливаться тенденция к сокращению операционных издержек, для того чтобы появилась возможность производить более разнообразные продукты. За этим последовала экспансия массового изготовления по индивидуальному заказу и 3D-печати. Методологии в сфере ИТ Agile и DevOps были направлены на достижение такого же эффекта.

Ранее операционные издержки были большими в ИТ-проектах. Была необходимость в приобретении оборудования и установке всякого рода межплатформенного ПО, а разработка оригинального продукта включала в себя множество задач, выполняемых вручную, таких как, например, развертывание изменений в коде, регрессионное тестирование данного решения, внесение изменений в окружение (так, для создания MVP тогда было необходимо потратить 6–12 месяцев вместо трех, а релиз новой функциональности занимал месяц вместо нескольких дней).

---

[1] Штефан Томке и Дональд Райнертсен, вероятно, лучше всех описали это в статье от мая 2012 года для Harward Business Review под названием *Six Myths of Product Development* («Шесть мифов о разработке продуктов»). И хотя они говорили непосредственно о разработке продуктов, а не об ИТ, так или иначе ИТ-услуги подразумевают поставку ИТ-продукта [1].

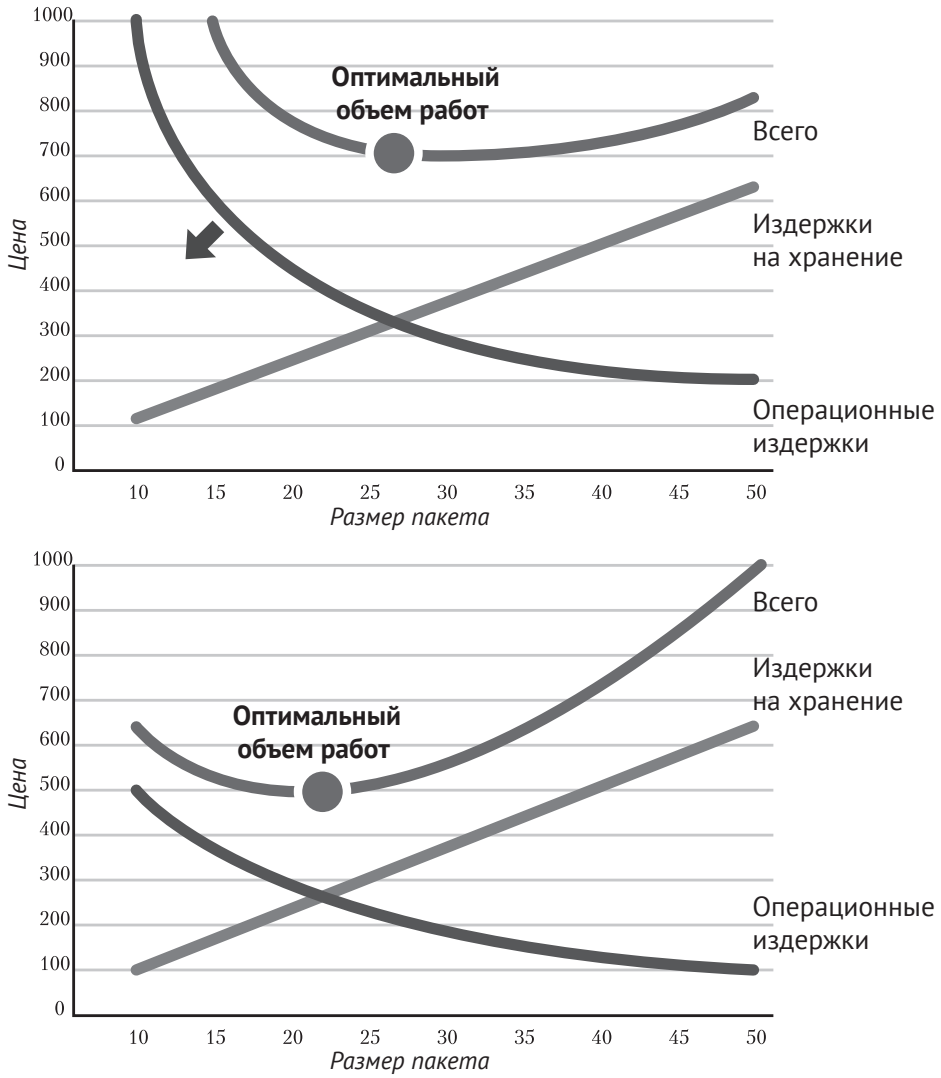


Рис. 3. Соотношение между операционными издержками и оптимальным объемом работ: уменьшение операционных издержек позволяет сократить объем работ

Сегодня это не так, и операционные издержки значительно снизились с началом применения облачных инфраструктур и XaaS и автоматизации процессов в жизненном цикле разработки, что позволило максимально сократить объем работ.

В этом малом оптимальном объеме работ таится настоящая выгода для бизнеса. Малый объем работ позволяет закладывать малые расходы на пробу бизнес-идеи и реже делать крупные ставки. Одну прекрасную аналогию этому по-

ложению высказал долгожитель в разработке продуктов, Дональд Райнертсен [2]: «Представьте себе лотерею, в которой вы получаете приз, если угадаете три цифры. Первый вариант ваших действий – дать мне пять долларов и попытаться угадать число из трех цифр, а затем я скажу вам, выиграли ли вы приз в $1000. Второй вариант – дать мне два доллара и попытаться угадать первую цифру. Я в свою очередь могу сказать вам, что эта цифра верная, а вы сможете решить, давать ли мне еще два доллара ради того, чтобы попытаться угадать следующую цифру. Какой из вариантов выбрали бы вы? На мой взгляд, второй вариант очевидно более привлекателен. Обратная связь при малом объеме работ позволяет получать информацию, на которой можно основывать решения о том, стоит ли идея дальнейших вложений. Это именно то, что обратная связь от заказчика позволит вам, если вы будете вносить небольшие партии новой функциональности и иметь дело с малым объемом работ в ваших ИТ-проектах».

Малый объем работ привлекателен еще и по другим причинам. Он упрощает предоставление ИТ-услуг, так как изменения производятся пошагово и в малом объеме, так что приходится меньше беспокоиться при тестировании и выпуске в среду эксплуатации. В производстве можно легко увеличивать объемы работ, так как вы можете выполнять один и тот же процесс в увеличенных объемах, не рискуя сильно усложнить его, как это может произойти при масштабировании вашего ИТ-решения. Меньший объем работ в ИТ позволяет нам проверить, в том ли направлении движется продукт, и пользоваться обратной связью от заказчика, чтобы направлять созидательную природу ИТ-разработки[1]. В производстве сложнее организовать циклы с проектированием, созданием и проверкой продукта: только представьте себе создание одного продукта на заводе (например, машины), его продажу и затем получение обратной связи, для того чтобы создать новый, и т. д. Задержки и расходы после запуска такого цикла вскоре начнут зашкаливать. Метод Lean-стартапа [3] Эрика Рая был адаптирован для продажи реальных товаров в технике Fast-Works, что позволило добиться замечательных результатов (уменьшение стоимости, увеличение скорости) [4] –по крайней мере, пока не стало нормой печатать все на 3D-принтерах и сокращать таким образом операционные и подготовительные расходы [5].

С малым объемом работ приходит совершенно другая экономическая модель для управления процессами с целью сокращения необходимых дополнительных издержек, требуемых от менеджеров. Другими словами, даже если технические операционные издержки были сокращены, это не значит, что автоматически снизятся и операционные расходы, связанные с менеджментом и обслуживанием архитектуры. А причина в том, что мы до сих пор следуем

[1] Так как в сфере ИТ свойственно совершать пробы и ошибки в процессе разработки продукта, меньший объем работ поможет сократить риски и ускорить получение обратной связи. Этим ИТ отличается от производства, в котором эффективность и продуктивность можно измерить.

принципам, перенятым из традиционного производства. Данная книга поможет вам отказаться от такого подхода.

В приложении я более детально рассмотрел некоторые идеи и принципы, которые перешли в сферу ИТ из производственной практики, и рассказываю о том, насколько они еще применимы сегодня.

Я надеюсь, что эта книга поможет вам создать предоставляющую ИТ-услуги организацию, которая не будет похожа на завод из фильма «Новые времена» с Чарли Чаплином: в ней люди будут работать вместе с заинтересованными сторонами, чтобы получить значимые для заказчиков решения. Как сказал мой коллега из Accenture Марк Рендел, «DevOps не стремится делать ИТ-услуги более эффективными. Речь идет об увеличении эффективности бизнеса с помощью ИТ» [6].

А теперь, осознав это, давайте приступим к преобразованиям.

ЧАСТЬ А

Создание подходящей экосистемы

Оказывая услуги консультанта, я часто говорю технологическим лидерам: представления о том, что такое хорошо, даже вкупе с благими намерениями, к сожалению, не всегда приносят наилучшие результаты. Экосистема, которую создает руководство компании, играет огромную роль в том, насколько успешными могут быть изменения. Скорее всего, экосистема, в которой вы работаете, была создана на фоне устаревших явлений – тех, которые были актуальны ранее, а также созданных ранее систем.

Понятие «устаревшие» применимо не только к вашим технологиям и приложениям. Как уже отмечалось во введении, существует консервативный тип мышления, вдохновленный командно-административной культурой, который желательно изменить. Итак, каким образом подготовить план, по которому будут проводиться изменения? Как выбрать правильные технологии? И кого же выбрать в надежные попутчики, чтобы уверенно двигаться от устаревших принципов к чему-то новому?

Владея инструментарием, предложенным в части А этой книги, вы сможете значительно приблизиться к созданию экосистемы, которая позволит ИТ процветать и преображаться. В этой части представлены высокоуровневые размышления о создания экосистемы, которая будет способствовать преобразованиям в современной ИТ-организации. Мы поговорим о пути преобразования, об ускорении работы в ИТ, о портфолио вашей компании, работе с устаревшими явлениями, выборе программного обеспечения и поиске подходящих партнеров, которые будут поставлять вам свои услуги. Также мы обсудим, как моя команда работает с клиентами – руководителями организации над изменением существующей экосистемы, стремясь к тому, чтобы это принесло пользу обеим нашим организациям; я покажу некоторые инструменты для анализа и действия, которые вы сможете создать самостоятельно. Иными словами, вы поймете, как работать с имеющимися в вашем распоряжении ресурсами так, чтобы они были для вас не помехой, а помощью.

ГЛАВА 1

Путь к изменениям

– Скажите, пожалуйста, куда мне отсюда идти?
– А куда ты хочешь попасть? – ответил Кот.
– Мне все равно… – сказала Алиса.
– Тогда все равно, куда и идти, – заметил Кот.

Льюис Кэрролл, «Алиса в Стране чудес»
(пер. Н. Демуровой)

Многие трудности в предоставлении ИТ-услуг связаны с узкими местами – этакими «бутылочными горлышками», которые не позволяют вам быстро получить значимый отзыв от клиентов, который вы смогли бы использовать для улучшения ваших рабочих процессов и продуктов. Сделать работу видимой – это один из наиболее действенных способов для обнаружения узких мест, однако ИТ чаще всего сталкивается с неявной работой: в ИТ нет акций, стоимость которых показывала бы успехи команды или компании в процессе создания продукта, нет никакого склада, по наполнению которого видно, каков объем невостребованной продукции, и нет никакого явного процесса, за которым мы могли бы наблюдать, чтобы увидеть, как то, что поступает на вход, превращается в конечный продукт на выходе. Это ведет к интересной ситуации: в то время как большинство людей, работающих на традиционном производстве, хотя бы приблизительно понимают, как их продукция создается, в ИТ процессы не так очевидны. Я имею в виду реальные процессы, а не те, которые могут быть описаны на сайте компании или же в какой-нибудь методологии. Однако без четкого представления о работе действительно трудно улучшить рабочие процессы. Таким образом, принципиально важная задача руководства любой ИТ-организации – сделать процессы видимыми, в том числе четко обозначить текущий статус работы и такие показатели, как качество и скорость. В этой главе мы воспользуемся картами потоков ценностей, чтобы достичь этого результата, и положим начало пре-

образованиям при помощи начальной карты потоков ценностей и управленческого подхода.

Явные ИТ-процессы

Я предпочитаю начинать свою работу в качестве консультанта по DevOps с упражнения по составлению карты потока ценности. Причина достаточно проста: это упражнение наиболее эффективно поясняет, как все же выглядят процессы в ИТ. Вы, конечно, можете изучить описание методологий или взглянуть на какие-нибудь большие диаграммы в Visio, чтобы узнать что-то про предоставление ИТ-услуг, но зачастую реальность не вполне совпадает с процессами, описанными в документации.

В конце каждой главы я коротко описываю процессы, которые вы можете попробовать воспроизвести, – смело пользуйтесь этим. В двух словах: вы организуете собрание с участием представителей каждого подразделения вашей организации, чтобы составить такой план предоставления ИТ-услуг, который всех устроит, и, что даже более важно, найти те элементы системы, которые можно улучшить. Предполагаю, что на такую встречу надо пригласить помощника или кого-либо, кто может судить о деле беспристрастно.

В идеале мы хотим иметь возможность объективно измерять такие параметры ИТ-процессов, как производительность, длительность циклов и качество. К сожалению, зачастую это требует немалых усилий. Составление карты потока ценностей раз в три–шесть месяцев (в зависимости от того, насколько быстро происходят изменения и улучшения) позволит вам отслеживать процессы, затрачивая на анализ их состояния всего несколько часов в месяц. Это даст вам возможность сосредоточиться на текущем процессе, на продолжительности циклов и качестве. Позаботьтесь о том, чтобы результаты составления карты были очевидными для всех сотрудников; тогда люди обратят внимание на улучшение процессов. Подобная практика послужит явным напоминанием о том, что улучшение процесса – важная задача для организации.

Когда вы уже сформируете в компании хорошее понимание высокоуровневых ИТ-процессов и способность видеть области, нуждающиеся в улучшении, настанет время разрабатывать первую схему трансформации.

Создание первой схемы трансформации

Создание схем – это отчасти наука, отчасти искусство. Многие схемы на первый взгляд похожи, но при детализации можно заметить, что у двух разных людей никогда не получаются одинаковые схемы. Тем не менее стоит отметить, что не существует идеальных схем. Поступая строго согласно принципам Agile, очень важно понимать общее направление и определять для себя некоторые опорные точки, чтобы осуществлять оценку процессов и делать их видимыми. Многие вещи со временем изменятся, и вам придется это учиты-

вать по ходу дела. Приведу рекомендации о том, как создавать хорошие схемы трансформации вашего предприятия.

Основываясь на карте потоков ценностей ваших ИТ-услуг, вы сможете определять узкие места в процессах. Как нас учат *системное мышление*, *теория ограничений* и *теория очередей*, пока мы не справимся с «бутылочными горлышками» в наших процессах, ни одно из нововведений не приведет к их ускорению. Это очень важный момент, поскольку иногда мы тратим время на мелочи, упуская то, что может вызвать глобальный сдвиг. Один из прекрасных способов выявления «бутылочных горлышек» – проведение практикума с картами потока ценностей. Позвольте заинтересованным сторонам заострить внимание на значимых проблемах, к которым нужно пристально присмотреться, чтобы изменить тактику предоставления ИТ-услуг в целом. Коллективные размышления в большинстве случаев помогают выявить ряд узких мест.

Чтобы создать успешно работающую схему трансформации, необходимо еще уделить время установлению баланса между процессами и скоростью предоставления ИТ-услуг и их стоимостью и качеством. Уделять большое внимание процессам – это яркий пример тактики, основанной на системном мышлении, которая может поспособствовать устранению барьеров в вашей организации. Ранее «ответственный» за выполнение функции, например за работу на определенном участке – допустим, с центром тестирования (Testing Center of Excellence) или фабрикой разработки (Development factory), – вел инициативы по совершенствованию, чтобы воздействие на эту область и контроль над ней были более эффективными. Со временем это поспособствовало возникновению функций для обслуживания большего объема работ, хорошо оптимизированных, но в ущерб предоставлению ИТ-услуг в целом. Процессы улучшаются только при малых объемах работ.

Обычно для оценки ИТ-услуг пользуются тремя показателями: скоростью, стоимостью и качеством работы. Ранее было принято прилагать усилия к улучшению показателей стоимости или качества, что, в свою очередь, сокращало скорость поставки услуг. Если вы будете оценивать свои ИТ-услуги только с опорой на улучшение качества, то зачастую будете наблюдать расширение границ показателей качества, что увеличит стоимость работ и время, затрачиваемое на их анализ. Если же будете оценивать выполняемую задачу, основываясь на сокращении показателей стоимости, то наиболее распространенный подход – доверить больше работы менее опытным сотрудникам или не выполнять отдельные задачи в рамках процесса, что часто приводит к ухудшению качества и замедлению работы из-за необходимости ее переделывать. Принимать во внимание лишь стоимость или качество, не учитывая влияния процессов, по моему опыту, неразумно: это не принесет успеха ИТ.

И наоборот, если нам нужна скорость (особенно это касается «бутылочных горлышек», которые мешают быстро осуществлять поставку действительно малых партий продукта), то, обращая внимание на процессы, мы сможем ускорить поставку продукта – даже больших его партий, – что приведет к улучшению показателей качества и стоимости работ. Невозможно добиться скорой

поставки, если с качеством дела обстоят неважно, так как необходимость переделывать некачественно выполненную работу замедлит процесс. Единственное, что вы можете сделать для ускорения поставки, – автоматизировать процессы и избавиться от ряда неактуальных шагов. Если просто научиться быстро печатать на клавиатуре, это не ускорит выполнение процессов в целом. Между тем ИТ вынуждена стремиться к увеличению скорости поставки услуг. Я участвовал в процессах трансформации вместе с клиентами, которые поработали над сокращением показателя стоимости ИТ-услуг, но в целом впечатление заинтересованных лиц о процессах поставки оставалось неудовлетворительным. Я также был свидетелем множества инициатив по улучшению качества, которые сдерживали процесс поставки и даже почти полностью останавливали его. И я еще буду встречаться с инициативами по совершенствованию, связанными с ускорением процессов.

И еще несколько слов в предостережение тем, кто работает над ускорением процессов. Первое, о чем хочется сказать, – не такая уж большая проблема. Вы на самом деле можете обойти проблему проведения оценки быстроты поставки, уменьшая объем поставляемых партий продукта, которые, в свою очередь, можно будет поставлять быстрее. И хотя это не значит, что вы сможете добиться взаимозаменяемости партий разных размеров, данный подход все равно окажется выигрышным для организации, поскольку менее объемные партии так или иначе несут в себе меньше риска. Второе предостережение: люди могут предложить способы сократить путь, которые окажутся рискованными или ухудшат качество продукта. Чтобы этого избежать, вам нужно продолжать поиски способов оценки качества, помимо скорости, чтобы убедиться в том, что качество продукта не ухудшается при ускорении выполнения процессов. Для оценивания скорости процессов вам будет необходимо обратить внимание на работу, осуществляемую на протяжении всего цикла предоставления услуг, а также на изучение показателей, которые проинформируют вас о статусе этих работ. Полезными показателями для оценки быстроты процессов являются длительность цикла выполнения процессов (длительность цикла = период времени с утверждения рабочего процесса до его выполнения и выпуска результатов в среду эксплуатации) и объем работ, поставляемых за определенный период времени.

На рис. 1.1 показано, что в вашей схеме преобразования наверняка будут ключевые точки, связанные с основными функциями и возможностями (например, с автоматизированным регрессионным тестированием, упрощенным представлением бизнес-кейса), что логично. Тем не менее стоит обратить внимание и на нечто другое, а именно на схему покрытия областей применения и технологий. В следующей главе я расскажу, как создать портфолио областей применения, позволяющее вам определить ряд возможностей, которые вы будете стремиться развивать в процессе трансформации. Ваша схема должна содержать в себе приоритизированные списки (часто называемые волнами) областей применения, потому как ни одна крупная организация не сможет

развивать абсолютно все возможные направления. И не должна. Ведь некоторые области применения не стоят усилий и затрат.

И еще пару слов о схеме трансформации: для того чтобы добиться больших возможностей и перемен, потребуется много времени. К сожалению, организации не очень справляются с тем, чтобы терпеливо ждать завершения таких программ, поэтому вам нужно убедиться в том, что вы можете предоставить какой-то видимый результат на раннем этапе. Для этих ранних результатов можно не придерживаться всех правил. В них можно реализовывать области применения, которые не критичны для бизнеса, или те, которые не являются частью «бутылочных горлышек». Эти ранние результаты необходимы для того, чтобы сохранить поддержку и позволить организации увидеть первые успехи. Вам стоит рассматривать эту задачу как часть действий, необходимых для трансформации. Конечно же, в идеальной ситуации ранние результаты будут затрагивать одну из предопределенных приоритетных областей.

Преобразование вашей организации займет время. На рис. 1.2 вы можете наблюдать шаблон схемы трансформации, который я часто использую в работе со своими клиентами. Паттерн, изображенный там, я довольно часто встречал. Он начинается с применения Agile, а затем приходит понимание того, что Agile без использования практик DevOps (таких как автоматизация тестирования, автоматизация развертывания и т. п.) не предоставит вам возможность ускорить процессы настолько, насколько вам это нужно. По мере внедрения DevOps ваша работа начнет ускоряться, вы начнете видеть ограничения организации и факторы, сдерживающие скорость работы, которые свойственны данной операционной модели. Для того чтобы преодолеть инерцию, потребуются значительные усилия организации и поддержка руководства. Не расстраивайтесь, если изменения не будут видны на следующее же утро.

Управление трансформацией

Как упоминалось ранее, схема – это очень важно, но без должного управления трансформацией вы не сможете добиться нужного результата. Довольно часто трансформация прекращается. Нет никакой возможности предвидеть все испытания, которые будут мешать продвижению, и без должного управления, которое поддерживало бы баланс между дисциплиной и гибкостью, процесс преобразований затормозится. Управление трансформацией позволяет сделать результаты видимыми, а также дает возможность влиять на них. Это отличается от обычного управления процессов предоставления ИТ-услуг, осуществляемого для ваших инициатив (например, в случае изменения плана технического наблюдения). На встрече совместно с рядом агентов и консультантов по трансформации на саммите DevOps Enterprise в 2015 году мы пытались определить, что необходимо для успешного внедрения DevOps. Все мы пользовались разными подходами и работали в разных организациях, но сходились в том, что отличает успешную организацию: способность постоянно совершенствоваться и успешно управлять этим процессом.

Рис. 1.1. Схема трансформации:
пример, показывающий волны приложений и возможностей

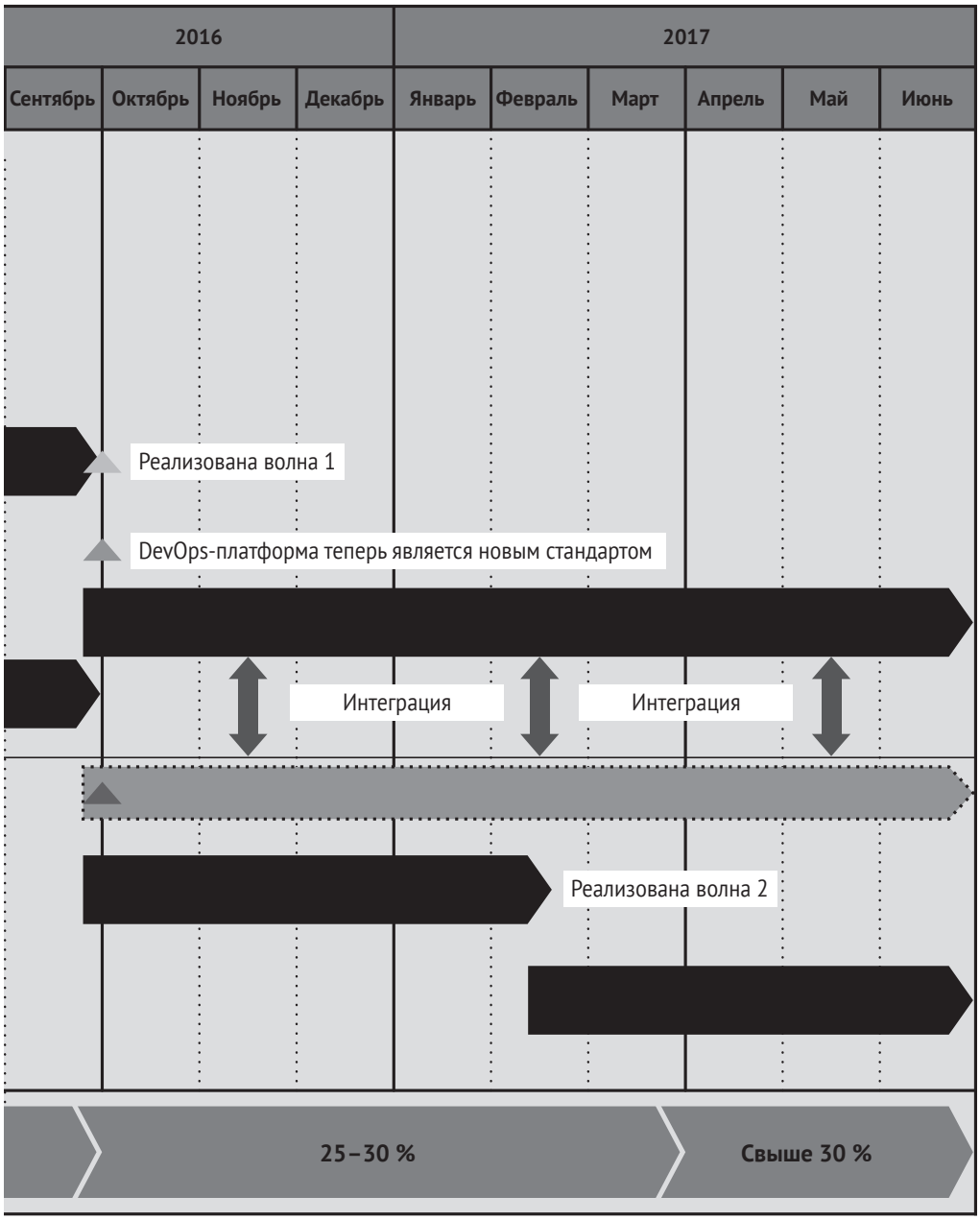


Рис. 1.1 (окончание)

Рис. 1.2. Распространенный шаблон схемы трансформации: изменение вашей организации займет время, пока вы будете пробовать различные методы

Рис. 1.2 (окончание)

Это непрерывное совершенствование и следование схеме трансформации вносит большой вклад в успешное преобразование вашей ИТ-организации. DevOps и Agile – не самоцель, поэтому не стоит стремиться к какому-либо конечному результату.

Как должно выглядеть успешное управление трансформацией? Управление трансформацией охватывает множество областей, поэтому важно, чтобы вы знали, с чем вам нужно сравнивать ваши успехи по мере реализации схемы. Это значит, что вам нужно определить точку отсчета перед началом преобразований, для того чтобы в дальнейшем оценивать показатели успешности. Слишком часто я встречался с инициативами по трансформации, когда шесть месяцев уходило на то, чтобы улучшить ситуацию, но затем нельзя было представить никаких доказательств тому – разве что анекдотичное «зато теперь у нас есть непрерывная интеграция с *Jenkins*». К сожалению, это не всегда убеждает руководство компании или другие заинтересованные стороны продолжать инвестировать в трансформацию. Даже если исполнительный директор будет за, компания не профинансирует дальнейшую работу по внедрению изменений: ведь никаких промежуточных результатов не достигнуто.

Тем не менее если вы сможете доказать, что с внедрением непрерывной интеграции вы смогли на 30 % снизить количество сбоев в работе запущенных экземпляров в окружении для сборки продукта, то у вас будут аргументы. Поэтому я настоятельно рекомендую выполнять упражнение по определению точки отправления в самом начале процесса трансформации. Подумайте обо всем, что вам важно, а также обо всем, что необходимо изменить; найдите подходящий способ определить текущее состояние. (Я привел несколько примеров в табл. 1.1.) Об использовании метрик я расскажу, когда мы немногим позже в этой главе будем говорить об управлении процессом предоставления ИТ-услуг.

Другой важный аспект управления трансформацией – поддержание гибкости и отслеживаемости работы. Для каждой инициативы по совершенствованию, согласно схеме, вам нужно применять научный подход:

- сформулируйте гипотезу, основанную на показателях успеха;
- составьте набор показателей;
- как только реализация будет готова, оцените результаты, используя гипотезу.

Что-то сработает, что-то – нет; в процессе управления трансформацией вам понадобится тот и другой опыт. Не вините проектную команду за неверную гипотезу (все-таки на их месте должны быть мы, руководители, которые заварили кашу, – кого здесь винить?). Тревогу стоит бить только в случае,когда сотрудники не следуют процессу (например, показатели не были предоставлены или результаты были искажены), что, в свою очередь, мешает вам двигаться дальше.

Таблица 1.1. Показатели базовой метрики, которые доказали свою эффективность в управлении трансформацией

Показатель	Определение	Измерение
Длительность цикла релиза	Среднее время, затрачиваемое на объем работ от утверждения рабочего пакета (пользовательской истории, функциональности, набора требований) и до его релиза	Обычно показатель измеряется как временная разница между моментами приобретения рабочим пакетом какого-либо состояния в вашей трекинговой системе
Стоимость релиза	Объем работы, необходимый для выпуска новой функциональности, измеряемый как усилия, затрачиваемые на всю деятельность для релиза (или же может учитываться только работа, совершаемая в нерабочее время)	Обычно данные основаны на табелях рабочего времени
Длительность регрессии	Время, требуемое для подтверждения того, что при внесении изменений не появились дефекты	Период с момента развертывания до «зеленого света» после автоматизированной или ручной валидации
Доступность в среде эксплуатации	Доля времени, в течение которого среда эксплуатации доступна для должного предоставления услуг	Измеряется как доля времени, в течение которого среда эксплуатации доступна для использования, или как доля успешно проведенных операций
Среднее время восстановления	Время, которое требуется для исправления дефектов в среде эксплуатации	Измеряется как время с момента возникновения дефекта до момента полного его исправления
Долговечность команд	Средняя продолжительность времени, в течение которого команды не распадаются	Время (в месяцах) до того момента, когда команды начинают распадаться и перенаправляться на другие проекты

Пока вы развиваетесь, развивается и ряд жизнеспособных инициатив по совершенствованию. Критерии оценки для инициатив, которым вы хотите дать ход, должны быть основаны на:

- более раннем опыте;
- размере инициативы в рамках системы WSJF (Weighted Shortest Job First);
- уровне лояльности: насколько команда готова постоять за инициативу.

Не смущайте себя тем, как обстоят дела у крупного бизнеса, которому на начальном этапе необходимы большие вливания. Вам стоит поначалу ставить небольшие задачи, чтобы проверить, насколько реализуема идея. Понадобится приглядывать за общей схемой, чтобы видеть и понимать, что цели, обозначенные в ней, вполне достижимы. Если это не так, вы можете изменить количество инициатив по совершенствованию или обновить схему (если этого никак не избежать).

В процессе управления трансформацией вам необходимо иметь некоторое представление обо всех частях организации, чтобы следить за тем, что изме-

нения не дают крен (например, в сторону тестирования, разработки или администрирования). Встречи, на которых будет обсуждаться процесс управления трансформацией, должны проводиться хотя бы раз в месяц, и необходимой документации для таких встреч должно быть как можно меньше. Если команда будет тратить кучу времени на то, чтобы оценивать презентации PowerPoint для каждой встречи, это не принесет пользы процессу трансформации. В идеальном случае для поиска путей к совершенствованию я бы посоветовал вам наблюдать за данными в реальном времени, за картой потоков ценностей и за несложными бизнес-кейсами.

Видимые ИТ-услуги

Если говорить о том, что можно визуализировать вещи при помощи реальных данных, стоит упомянуть, что некоторые возможности DevOps могут весьма и весьма для этого пригодиться. Одним из лучших помощников в таком деле может оказаться конвейер развертывания[1]. Конвейер развертывания –

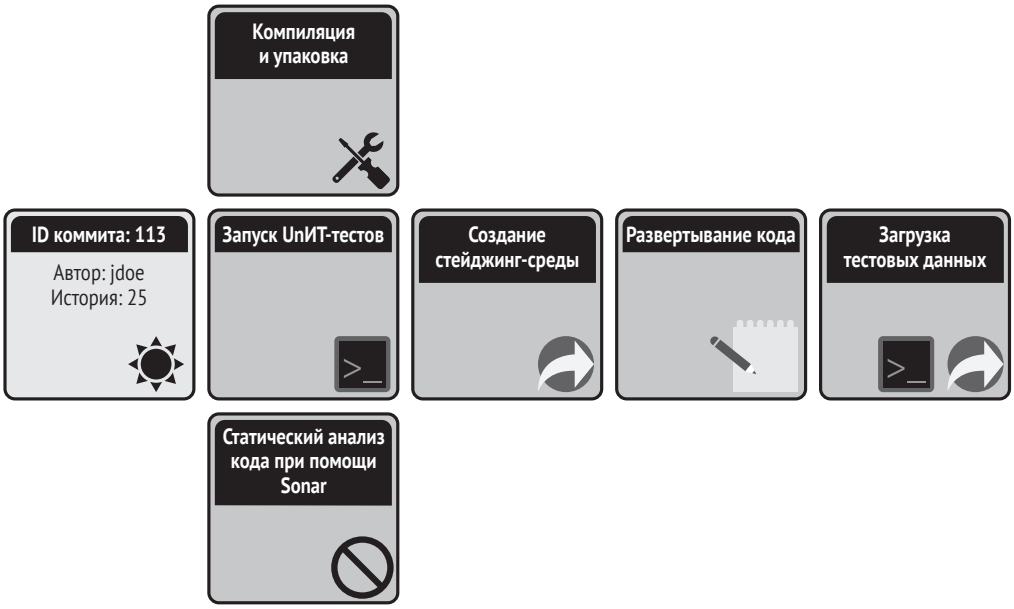


Рис. 1.3. Пример конвейера развертывания:
DevOps-платформа Accenture предоставляет возможность наблюдать
за процессом развертывания

[1] Гэри Грувер написал целую книгу, *Starting and Scaling DevOps in the Enterprise*, о конвейере развертывания как о средстве, способствующем трансформации.

это визуальное представление процесса, в результате которого программный продукт проходит путь от разработчика к среде эксплуатации с определенными промежуточными стадиями. Это визуальное представление показывает, что происходит с программой, а также любой положительный или отрицательный результат ее работы. На рис. 1.3 я оставил пример одного решения, с которым часто работаю. Вы можете видеть, как различные стадии жизненного цикла и связанных с ними задач представлены в виде конвейера [1]. Этот конвейер развертывания непосредственно предоставляет информацию о качестве вашего программного продукта в реальном времени. Вы можете также подумать о том, чтобы агрегировать дополнительную информацию на панели мониторинга (или дашборда) либо скомбинировать основные данные с дополнительными, но так или иначе конвейер развертывания предоставляет некоторый фундамент. Он несет в себе вместе с тем и действенную функцию, так как все его шаги имеют некоторое представление и обязательно исполняются, а результаты можно просмотреть непосредственно на панели мониторинга (или дашборда) – это снизит вероятность того, что люди начнут осуществлять неотслеживаемые действия. Любые улучшения и изменения процесса можно видеть в конвейере развертывания до тех пор, пока это единственный разрешенный способ поставки изменений. Если доступ к метрикам затруднен, вы можете добавить шаги на каждом этапе конвейера, для того чтобы логировать метрики, необходимые для дальнейшего анализа.

Рис. 1.3 (продолжение)

Сегодня важно иметь в своей компании аналитическое решение для создания панелей мониторинга за данными в реальном времени. Многие компании пользуются готовыми решениями для визуализации или аналитики либо же создают нечто на основе open-source вариантов (наподобие Graphite). Основной целью здесь является использование данных, генерируемых на протяжении всего жизненного цикла, чтобы наполнять панели мониторинга (или дашборды), которые можно будет использовать не только для целей управления трансформацией, но и в любом подходящем случае. Высокопроизводительные команды связали цепочку DevOps-инструментов с аналитическими панелями мониторинга, и это позволяет им видеть важную информацию в реальном времени. Например, мы можем наблюдать за качеством текущего релиза, за тем, как качество выпускаемого пакета связано с проблемами, возникающими после развертывания, а также за тем, насколько автоматизация тестирования улучшила показатели по дефектам на более поздних стадиях жизненного цикла продукта.

Управление процессом поставки ИТ-услуг

Управление ИТ-процессами, по-моему, недооценено как составляющая часть процесса трансформации. Правда в том, что большинство подходов к управлению имеют существенные недостатки и помогают достичь лишь малой доли запланированных результатов. На большинстве встреч по управлению, на которых я присутствовал или в которых принимал участие, использовались текущие отчеты, составленные по системе «светофора», которые я считаю довольно субъективными по своей природе, и они не совсем подходят для представления статуса работы. Более того, даже если критерии для цветовой гаммы где-либо прописаны, зачастую руководство предпочитает узнавать личное мнение проектного менеджера. Проектные менеджеры из *Института проектного менеджмента* (PMI) используют *показатель эффективности стоимости* (CPI) и *показатель эффективности рабочего графика* (SPI); они немного лучше, но все еще основываются на необходимости иметь подробный, правильно составленный проектный план, в соответствии с которым будет готовиться отчетность. Я придерживаюсь того мнения, что, поскольку большинство проектов со временем меняются, составлять подробный план всего проекта в целом – напрасная затея: вы наверняка обнаружите, что в какой-то момент отклонились от курса.

Кроме того, к тому времени, как текущий отчет будет представлен на встрече, он будет существовать уже как минимум пару часов. В худшем случае вы будете дезинформированы из-за того, что было упущено множество различных сообщений, а проектному менеджеру пришлось работать с не вполне корректными результатами. Слишком часто текущие отчеты пестрят зеленым цветом на протяжении многих недель и внезапно «краснеют», когда плохие новости уже невозможно игнорировать. Или, например, оцениваемое состояние под-

нимается по цепочке управления и все чаще и чаще маркируется зеленым цветом по мере продвижения вверх по списку, ибо все стараются показать, что владеют ситуацией. Помните: одна из наших целей – сделать работу видимой, и мы не сможем добиться значительных успехов, если представляемая нами информация будет далека от реальности.

Что имеет смысл использовать в управлении доставкой ИТ-услуг, так это объективные показатели, например: число работающих элементов функционала, время восстановления после инцидента, длительность циклов для доставляемой функциональности, а также истории/функциональность, поставляемые и принимаемые после каждой итерации. Эти показатели предоставляют достойные способы оценивать успехи и качество. Данную информацию, а также другие метрики не стоит собирать вручную. Вы должны иметь возможность получать информацию из панели мониторинга (или дашборда) в режиме реального времени. Некоторые метрики можно брать из вашего конвейера доставки, но для многих из них потребуются дополнительные источники данных (например, ваш инструмент для управления жизненным циклом Agile). «Цветной комментарий» предоставляется проектной командой, и его можно накладывать на презентацию, так как обсуждение или аннотированные скриншоты можно создавать для встречи, на которой будет обсуждаться управление (см. пример аннотированного графика выполнения работ на рис. 1.4).

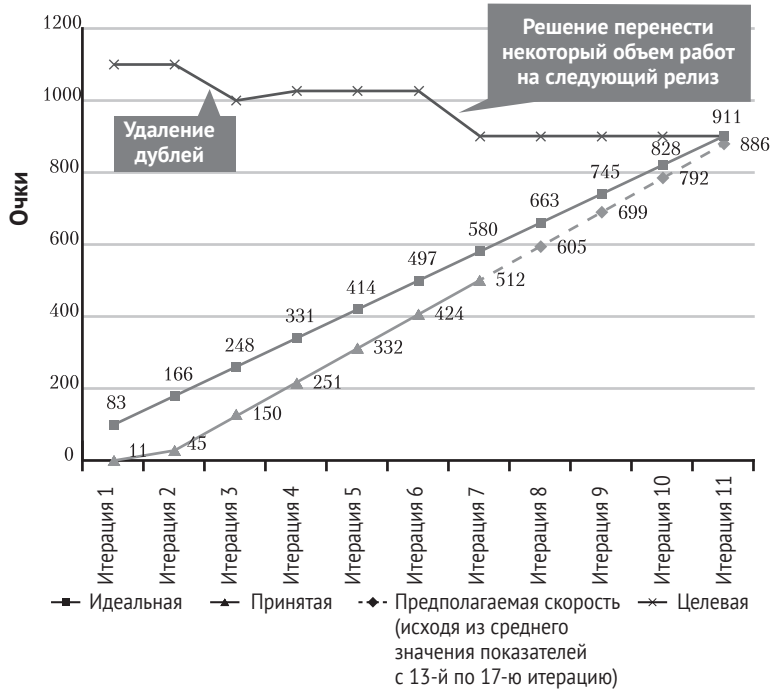


Рис. 1.4. Аннотированный график выполнения работ: на графике отображены аннотации статуса проекта

То же самое справедливо и для метрик, которые вы используете для управления ИТ-услугами. Нет весомых причин для того, чтобы не иметь таких данных по каждой стадии вашего процесса предоставления ИТ-услуг. Меня удивляет, что мы пользуемся ИТ для того, чтобы создавать замечательные аналитические решения для бизнеса, но сами же эти мощные решения не применяем в целях улучшения процессов организации. Ручной сбор данных для метрик неприемлем, если есть возможность внедрить в ваш процесс шаги по автоматическому логированию. В худшем случае вы немного автоматизируете каждый из шагов, который выводит данные в распространенном для логов формате. Мне часто приводилось такое делать, поскольку большинство используемых инструментов в жизненном цикле продукта не выводили извлеченные данные в формате, который можно было бы легко использовать. Использование большинства инструментов предполагает применение встроенной функциональности по составлению отчетов, что не всегда подходит. Вам нужно иметь возможность связывать данные из одного инструмента с другим (и, возможно, с инструментами различных поставщиков), поэтому вам придется создавать свои инструменты, чтобы убедиться, что данные в жизненном цикле вашего продукта остаются доступными для анализа. Подобное внимание к данным со временем значительно окупится.

Такой большой объем данных запросто может оказаться непосильным, и объем данных, генерируемых в процессе поставки ИТ-услуг и администрирования, чрезвычайно велик, так что подходы к работе с ними можно сравнить с подходами, применяемыми к большим данным. Самое главное здесь будет сосредоточиться на узких «бутылочных горлышках». Пробуйте создать метрики, которые смогут описывать их состояние, и следите за ними в процессе трансформации. Когда вы справитесь с основной проблемой, акценты сместятся на другие, и тогда вам уже понадобятся новые метрики. К счастью, если создать хороший фреймворк для анализа метрик, включая панели мониторинга (или дашборды) и подготовку данных, то у вас на руках будет вся необходимая информация. Панели мониторинга (или дашборды), как описывалось ранее, – прекрасный инструмент, позволяющий агрегировать информацию и сделать ее доступной для других инструментов.

Предоставление ИТ-услуг по принципу Lean

В процессе трансформации внимание часто уделяется техническим практикам, хотя многое можно улучшить, если применять принципы Lean к подходам по управлению ИТ-услугами. Под управлением ИТ-услугами я подразумеваю любой этап в ходе ИТ-поставки, когда необходимо подтвердить нечто перед тем, как продвигаться дальше. Это могут быть ключевые точки по финансированию проекта, принятие решения о развертывании в тестовой среде, смена панели управления и т. п. В жизненном цикле разработки продукта обычно бывает множество таких шагов, требующих подтверждения или проверки, ко-

торые отнимают довольно много времени и сил. А процессы управления со временем разрастаются. После возникновения проблемы мы совершаем проверку реализованного функционала и добавляем еще один шаг, для того чтобы данная проблема больше не возникала. В конце концов, никому не повредит лишний раз убедиться дважды, правда? Со временем это спровоцирует появление процесса управления, шаги которого обессмыслятся, и отследить их станет нереально. Я наблюдал процессы подтверждения развертывания, на которые уходило значительно больше времени, чем обычное развертывание, не подразумевающее добавления ценности и улучшения качества. Мне кажется, что некоторые шаги подтверждения превратились в этакие управленческие шаблоны, не несущие особого смысла, так как информация сейчас не оценивается должным образом. Следующие рассуждения помогут вам сократить ненужные шаги в ваших процессах.

Я хотел бы, чтобы вы внимательно оценили каждый этап в вашем процессе управления, чтобы понять: 1) насколько этот этап важен (например, в контексте отклонения некоторой заявки); 2) насколько рискованно его проходить и 3) сколько это стоит.

Давайте рассмотрим каждый из этих трех аспектов в отдельности.

1. Если обратить внимание на шаги рассмотрения заявок в жизненном цикле разработки ПО, как часто заявки отклоняются и как часто при рассмотрении обнаруживаются реальные проблемы, которые требуют решения? (Я имею в виду действительно важные проблемы, не просто отказы по формальным причинам вроде неправильного заполнения бланка заявки.) Чем реже обнаруживается, что процесс приносит должные результаты, тем более вероятно, что он не несет в себе ценности. То же самое можно сказать в случаях, когда заявки одобряются с вероятностью 90 %. Наверное, стоит обойтись уведомлением, вместо того чтобы заставлять людей ждать рассмотрения заявки, которая с высокой вероятностью будет одобрена. Или же вы можете совершенно избавиться от этого шага. Я работал с одним клиентом, у которого сотрудники команды разработки были вынуждены просить одобрения заявок перед каждым развертыванием после того, как все подготовительные шаги были осуществлены, что увеличивало этап подготовки перед развертыванием. Рассматривающий заявку не делал никакой полезной работы, если учесть, насколько быстро принималось решение. Это была сущая формальность. Я порекомендовал избавиться от этого шага и изменить процесс так, чтобы информация просто отправлялась на рассмотрение до и после развертывания; то же самое касалось и результатов тестирования. Подготовительное время значительно сократилось, у сотрудника уменьшился объем работы, и из-за того, что мы убрали шаг, выполняемый вручную, у нас появилась возможность автоматизировать процесс развертывания с самого начала.
2. Если мы и далее будем выполнять процесс без этого шага и что-то пойдет не так, насколько большими будут риски? Как много времени понадобится,

чтобы выявить проблему и исправить ее или отказаться от изменений? Если риск небольшой, то, опять же, шаг можно пропустить или ограничиться отправкой уведомлений.
3. Сколько усилий и времени требуется на выполнение этого шага? Как долго готовится документация на данном этапе? Сколько времени тратит на этот шаг каждая из заинтересованных сторон? Сколько времени в жизненном цикле уходит на ожидание одобрения?

Имея на руках такую информацию, вы сможете оценить, нужно ли продолжать использовать этот шаг или его стоит убрать либо как-то упростить. По моему опыту, около половины шагов, подразумевающих рассмотрение заявок, можно или автоматизировать (так как человек при этом следует повторяемым шагам), или свести их к отправке уведомления, в результате чего процесс перестанет замедляться. Я советую вам попробовать этот подход в вашей организации. Вы увидите, как легко на самом деле отказаться от всего ненужного и приблизиться к достижению минимального жизнеспособного процесса управления. Упражнение на такие случаи приведено в конце этой главы.

Первые шаги вашей организации

Здесь я представлю три упражнения, которые считаю невероятно полезными, так как они помогают достичь значительных результатов относительно малыми силами: 1) составление карты потока ценностей для вашего процесса поставки ИТ-услуг; 2) формирование основных метрик и 3) пересмотр подходов к управлению процессами. В результате вы сможете лучше понять ИТ-процессы и начать совершенствоваться.

Составление карты потоков ценностей для ИТ-процессов

Существует формальный процесс для составления карты потоков ценностей, однако здесь я представлю сокращенную версию, которая, по моему опыту, сравнительно неплохо помогает достичь поставленной нами цели: сделать процессы видимыми и справиться с некоторыми узкими местами[1]. Вот этот краткий план по составлению карты потоков ценностей:

1. Соберите на переговоры всех заинтересованных участников цепочки выноса поставок (supply chain): например, представителей бизнеса, сотрудников отделов разработки, тестирования, менеджеров проекта, администраторов и бизнес-аналитиков).
2. Подготовьте высокоуровневую схему процесса. Стоит для наглядности упомянуть на ней такие понятия, как, например, «бизнес-идея», «старт проекта», «разработка», «тестирование/QA», «развертывание/релиз» и «создание ценности».

[1] Вам стоит обратиться к Value Stream Mapping Карен Мартин и Майка Остерлинга, если нужно в большей степени формализировать процесс.

3. Попросите всех за 15 минут совместно записать краткое пошаговое описание ИТ-процесса на карточках. Затем попросите приклеить эти карточки на доску, чтобы общими усилиями составить полноценную картину процесса. Внимание: вам, возможно, понадобится воодушевлять людей на сотрудничество или самому вступить в обсуждение, если ситуация будет развиваться в нежелательном направлении.
4. Как только карта будет составлена, попросите нескольких человек заново описать весть процесс, а также спросите, стоит ли что-то добавить.
5. Теперь, когда у вас есть осмысленное представление процесса, вы можете пытаться подробно обсудить его циклы, ключевые точки представлений заинтересованных сторон о качестве или каких-либо других аспектах, а также инструменты, которые будут помогать обслуживать процесс.
6. Попросите людей проголосовать за наиболее важные узкие места (например, дайте возможность каждому оставить лишь три отметки на доске напротив трех самых значимых в этом отношении шагов).

По моему опыту, это упражнение наилучшим образом помогает сделать ваш процесс видимым. Вы можете повторять данное упражнение каждые три–шесть месяцев, чтобы понимать, правильно ли вы обращаетесь с узкими местами, и видеть, как развивается процесс. Можете наглядно представитьрезультаты этого процесса, вывесив их где-нибудь в офисе, чтобы показать всем приоритетные направления совершенствования. Узкие места, которым было уделено наибольшее внимание на встрече, могут стать ключевыми точками в вашей начальной схеме, поскольку это то, на что должны ссылаться ваши инициативы.

Формирование основных метрик

Так как наличие метрик невероятно важно для процесса управления трансформацией, я хочу, чтобы вы посвятили несколько минут заполнению табл. 1.2. Выделите для себя метрики, которые важны для вас сейчас и в будущем, и определите механизм, согласно которому вы будете их составлять. Существует несколько способов формирования основного ряда метрик. Они могут быть основаны на опросах, текущих исследованиях или, в идеальном случае, на существующих либо исторических данных. Если задача чересчур сложна, то вам стоит подумать об автоматизированном способе измерения метрик. Там, где это не сработает, вы можете проводить исследование вручную и оценивать процесс самостоятельно (например, во время текущих исследований), но такие метрики будут вызывать меньше доверия и их составление потребует больше времени.

Таблица 1.2. Пример формирования метрик: метрики должны иметь описание, механизмы составления и базовое значение

Метрика	Описание	Механизм составления	Основной (базовый) подход	Базовое значение
Длительность цикла	Среднее время, необходимое для того, чтобы продвинуться от готовой истории до развертывания в среде эксплуатации	Извлечение даты и времени из Agile-системы управления жизненным циклом	Анализ пользовательских историй, которые были успешно развернуты в среде эксплуатации за прошедшие шесть месяцев	168 дней

Пересмотр подходов к управлению процессами

Многое можно обсудить в части автоматизации, которая способна помочь усовершенствовать скорость и качество ИТ-процессов. Что люди часто недооценивают, так это то, каких результатов можно добиться за счет одной лишь коррекции процесса управления. Ниже я привожу небольшой список вопросов, который вы можете использовать для того, чтобы пересматривать ваш процесс. Задавайте эти вопросы, чтобы сосредоточиться на тех элементах, где управление ИТ-процессами действительно необходимо. Ваши ответы позволят оценить эффект и риски от сокращения шага в процессе, в идеальном случае даже с применением экономической модели, отражающей финансовое воздействие и сформированной вероятностью рисков.

Список вопросов об управлении процессом:

- Как часто кто-либо отклонял заявку на выполнение одной из задач, основываясь на причинах, не связанных с правилами соответствия процессу?
- Что может произойти с процессом, если будет совершен неверный выбор?
- Какую ценность принесет человек, одобряя эту заявку вручную вместо компьютера, который может осуществить этот шаг при помощи ряда параметров?
- Как много времени и средств будет затрачено на управление процессом (включая обычное время ожидания одобрения заявки)?
- Основан ли этот шаг на объективных показателях или на субъективных? Как вы это определили?

ГЛАВА 2

Принятие быстро меняющейся реальности

> Если все кажется одинаково важным, значит, ничего из этого таковым не является.
>
> *Анонимный автор*

У клиентов, с которыми я работал, были тысячи приложений в портфолио. Вполне очевидно, что мы не можем вносить изменения в каждое из них одновременно. В этой главе мы посмотрим, как ориентироваться в мире новых инновационных систем и устаревших приложений. Мы определим *минимальные жизнеспособные кластеры* приложений, для того чтобы приступить к трансформации управления, и для этого проанализируем портфолио.

Одной из тенденций, обсуловившей возрастание интереса к практикам Agile и DevOps, в ИТ-индустрии стал приход интернет-поколения, о котором я говорил во введении. Компании, у которых приложения новее, чем большинство приложений в мире крупных корпораций, имеют большое преимущество. Слово «устаревший» часто используется в нашей индустрии с негативным подтекстом, но правда в том, что любой код, развернутый в среде эксплуатации, уже устарел. И любой новый код, который мы напишем сегодня, завтра уже устареет. Пытаться отличить устаревший код от неустаревшего со временем станет еще труднее.

В прошлом организации пытались справиться с устаревшими приложениями с помощью проектов по трансформации, которые занимали много лет, и заменить устаревшие системы новыми. Но довольно часто старые системы выживали по той или другой причине, и архитектура приложения в целом становилась все сложнее. Больше нет тенденции заниматься радикальными пре-

образованиями, так как для ускорения эволюции организациям необходимо уметь подстраиваться в процессе изменения их ИТ-архитектуры.

Думаю, вы согласитесь с тем, что мы все хотим выработать действительно быструю, гибкую и надежную тактику предоставления ИТ-услуг. Должны ли мы при этом избавляться от устаревших приложений и создавать ряд быстрых приложений? Полагаю, что реальный мир не так прост. Я работал с десятками организаций, которые метались между быстрыми цифровыми приложениями и медленными приложениями крупных компаний. Некоторые из этих организаций, только завершившие большой процесс трансформации, который должен был решить эту проблему, уже имели новые приложения, которые к концу трансформации стали медленными и устарели. Нужен был более практичный подход, который принесет результаты.

В то время как нам необходимо, чтобы все работало быстро, придется смириться с тем, что архитектура некоторых приложений и накопленный *технический долг*[1] могут не позволить поставлять каждое из приложений с одинаковой скоростью. Сегодня ведутся дискуссии о бимодальных ИТ (использующих два метода предоставления ИТ-услуг, таких как водопад для предсказуемости и Agile для экспериментов) [1] или о мультимодальных ИТ (использующих различные методы, такие как Agile и разновидности водопада), в которых приложения разделяются на типы (*системы взаимодействия* для общения с клиентами и *системы учета* для внутренних процессов) [2]. Я думаю, что эта сложная классификация в некотором роде может помешать, если вам нужно добиться большей скорости; если ваш бизнес полагается на системы учета для разграничения обязанностей, то такие системы должны поставляться настолько быстро и надежно, насколько это возможно. Многие организации используют эту классификацию в качестве аргумента для того, чтобы не улучшать некоторые приложения, но с точки зрения бизнес-ценности это неверно.

В этой главе я представлю альтернативный метод, который использую в работе с моими клиентами, чтобы помочь им сформировать быстро развивающийся подход, благодаря которому все будет работать настолько быстро, насколько позволяют реализуемость и экономическая целесообразность. Это может привести к различающейся скорости поставки продуктов в будущем.

---

[1] Я немного расскажу о техническом долге в этой главе, поэтому хотел бы предложить вам подумать о том, как его можно измерять. Существуют, конечно, инструменты для анализа кода, которые позволяют получить некоторое представление о техническом долге, основанное на проблемах кода. Пожалуй, они для начала вполне сгодятся. Вместе с тем я бы использовал стоимость развертывания приложения (например, как много человеко-часов необходимо для развертывания продукта в тестовой среде или в среде эксплуатации), стоимость регрессионного тестирования продукта (как много человеко-часов необходимо для того, чтобы проверить, что ничего не сломалось) и стоимость создания новой среды с приложением. Если вы готовы пойти еще дальше, то можете взглянуть на показатели сложности и зависимости от других приложений, но я еще не встречал хорошего воспроизводимого способа измерить их. Другие же четыре показателя относительно несложно определить, и, таким образом, они должны лечь в основу измерения технического долга.

Анализируем портфолио приложений

У крупных организаций часто имеются сотни, если не тысячи приложений, поэтому было бы слишком самонадеянно предполагать, что мы можем усовершенствовать все приложения разом. Некоторые приложения и не надо улучшать, так как они довольно редко изменяются и не представляют стратегической ценности. В заключительном разделе главы, где приводится упражнение, я покажу подробнее, как вы можете осуществлять анализ самостоятельно.

При помощи такого анализа мы можем делать несколько вещей: приоритизировать приложения по кластерам (я расскажу об этом немного позднее) и распределить все приложения по трем группам, которые помогут определить, как обращаться с каждым приложением по мере осуществления трансформации. От того, к какой группе относится приложение, будет зависеть, как вы будете выделять на него ресурсы и как будете работать с поставщиками программных продуктов и вашими партнерами по предоставлению ИТ-услуг.

В первую группу мы поместим приложения, от которых собираемся избавляться или которые не собираемся часто изменять. Давайте назовем их *истинно устаревшими*, чтобы отличать их от «просто устаревших», то есть просто от более старых систем. В категорию истинно устаревших попадают приложения, которые весьма редко изменяются, которые не обслуживают процессы, важные для бизнеса, и в которые вы не вкладываете ресурсы. Вполне очевидно, что вам не хочется тратить много средств на автоматизацию жизненного цикла таких приложений. Ради их обслуживания вы вряд ли будете тратить много времени на выбор поставщиков и предпочтете партнера, который обеспечит базовую работоспособность по низкой цене, если не желаете поручать работу с такими приложениями штатному сотруднику. И вам действительно не стоит уделять этому вопросу слишком много внимания.

Во вторую группу мы поместим приложения, которые обслуживают ваш бизнес, но немного далеки от ваших клиентов. Вспомните о ERP- или HCM-системах – это «рабочие лошадки» для ваших приложений. Вы тратите кучу денег на запуск и поддержку этих систем, и они, вероятнее всего, определяют скорость поставки в крупных проектах. Занимаясь этими «рабочими лошадками», вы сможете позволить себе поставлять продукт более быстрым и надежным способом, но технологии, по которым они работают, не так легко адаптировать под Agile и DevOps. При работе с такими системами крайне важно сотрудничество с поставщиками, чтобы эти технологии могли лучше сочетаться с DevOps (мы поговорим об этом в следующей главе). Если вы все-таки решите поддерживать и развивать эти системы, убедитесь в том, что ваш партнер разделяет ваше желание совершенствовать способы работы и саму систему.

В третью группу войдут приложения, которые будут выступать «двигателями инноваций». Это приложения, обращенные к клиентам, которые вы можете использовать для продвижения инноваций, но, с другой стороны, они могут и навредить, если клиентам не понравится то, что вы им представляете. Проб-

лемы возникнут, если большинство из них будет полагаться на тех «рабочих лошадок», которые помогают создать правильные ощущения от продукта. Мой любимый пример – банковское мобильное приложение, которым вы можете спокойно пользоваться, пока оно отображает правильную информацию о ваших банковских счетах; в противном случае вы будете весьма расстроены как клиент. Для этой группы приложений вы будете использовать оригинальные технологии. Вам стоит плотно работать с вашим поставщиком программных продуктов, если вы решите использовать готовые коммерческие компоненты (COTS), и партнер по поставке должен быть участником созидательного процесса, а не просто партнером по предоставлению ИТ-услуг.

Рис. 2.1. Радар приложений:
визуализирует статус каждого приложения

И на самом деле эти группы приложений не статичны. По мере развития архитектуры ваших приложений некоторые из них переместятся в другие группы, и сообразно этому будет развиваться ваша стратегия выбора поставщиков и партнеров. Действенное управление портфолио приложений становится все более важным аспектом, так как скорость развития увеличивается и архитектура приложений становится все более модульной. Продолжая начатый в пер-

вой главе разговор о том, что необходимо делать процессы видимыми, скажу, что наилучшим способом представления проведенного анализа будет радар приложений с группой для «двигателей инноваций» с самой его середины и группами для «рабочих лошадок» и истинно устаревших приложений, соответственно, в двух внешних кругах.

Поиск минимального жизнеспособного кластера

Принцип Agile о партиях с малым объемом работ применим также и к процессу трансформации. Для руководства мы можем использовать информацию из анализа портфолио приложений. Весьма вероятно, что приложения из категорий с «рабочими лошадками» и «двигателями инноваций» часто будут работать одновременно. Вместо того чтобы брать первые несколько приложений, вам нужно сначала провести анализ, который позволит найти то, что я называю минимальным жизнеспособным кластером.

Приложения не живут отдельно друг от друга. Следовательно, большинство функциональных изменений в ландшафте вашего приложения потребуют от вас обновления более чем одного приложения. А следовательно, даже если вы способны ускорить работу приложения, это не значит, что вы сможете ускорить поставку: ведь вам все еще придется ждать, пока в другие приложения будут внесены изменения.

Здесь напрашивается аналогия со слабым звеном. В нашем случае слабое звено определяет, с какой скоростью будет происходить поставка. Что вам нужно определить, так это минимальный жизнеспособный кластер приложений. Для этого лучше всего оценить ваши приложения по нескольким факторам, таким как клиентоориентированность и частота и объемы внесения изменений. Идея нахождения минимального жизнеспособного кластера состоит в том, что вы с некоторой периодичностью пересматриваете высокоприоритетное приложение и анализируете его зависимости. Вам нужно выделить небольшой ряд приложений, в котором вы сможете замечать увеличение скорости поставки с увеличением скорости поставки каждого приложения в этом ряду. (Иногда придется иметь дело и с дополнительными зависимостями, но в большинстве случаев составление данного ряда приложений позволит вам независимо вносить значительные изменения, проявляя некоторую креативность.)

Вы можете не останавливаться и проводить анализ и далее, формируя другие кластеры, чтобы получить лучшее представление о приложениях, к которым вы будете обращаться в дальнейшем. Не тратьте слишком много времени на кластеризацию всех приложений. По мере продвижения вы сможете волнообразным способом определять кластеры.

Отдельно нужно сказать несколько слов по поводу приоритизации приложений. В первую очередь я считаю, что важно начинать работу над критичными приложениями как можно раньше. Многие организации экспериментируют

в изолированных приложениях с новыми техниками для автоматизации, при условии что это не окажет значительного воздействия на бизнес. Многие техники, которые работают в таких приложениях, не получается масштабировать и распространить на весь остальной ИТ-ландшафт, и вся организация не видит значительной перемены в этом приложении. («Они не работают в наших реальных системах», – комментарий, который можно услышать в такой ситуации.)

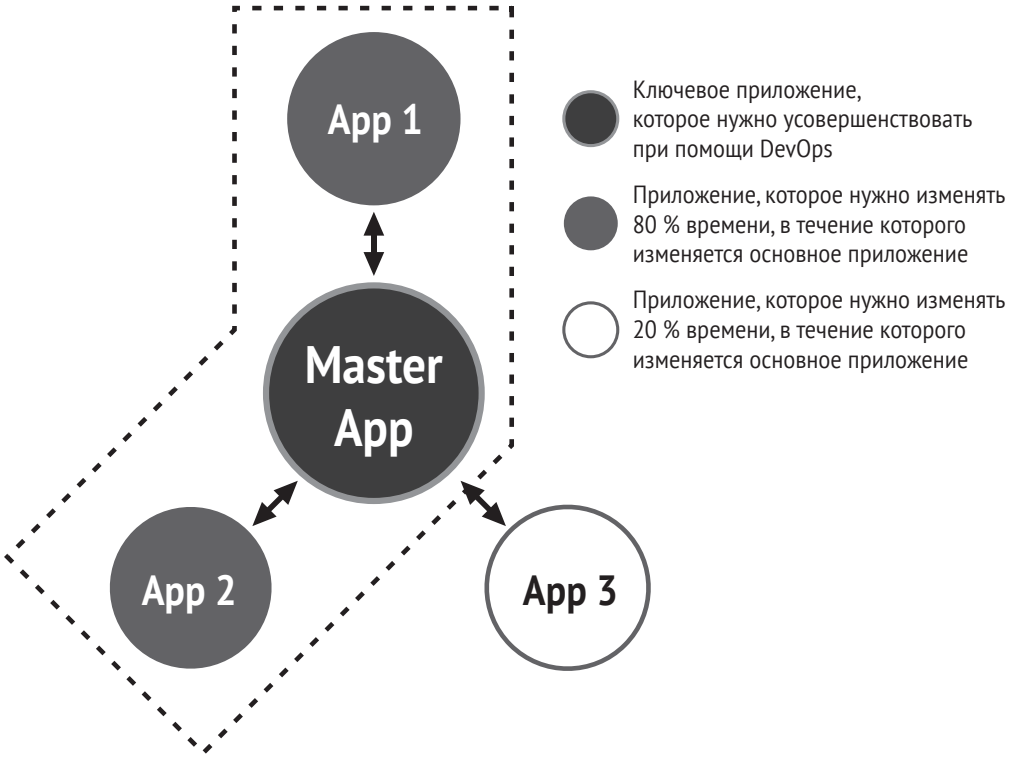


Рис. 2.2. Минимально жизнеспособный кластер: применение системного мышления в анализе приложений

Поскольку формирование минимального жизнеспособного кластера может потребовать от вас времени, стоит попытаться найти какое-либо руководство для того, чтобы можно было: а) как можно раньше предоставить результат и б) иметь возможность узнать о более продвинутых техниках, прежде чем начать составлять ваш первый минимальный жизнеспособный кластер. Чтобы это сделать, вам нужно убедиться в том, что выводы о минимальном жизнеспособном кластере подтверждаются на примере более простого приложения; так организация сможет увидеть целесообразность вашей идеи. Согласованная работа заинтересованных сторон различных приложений крайне важна для достижения такого результата.

Что делать с истинно устаревшими приложениями

Мы поговорили о стратегии, которую вам необходимо применять к приложениям, остающимся частью вашего портфолио, но что же делать с истинно устаревшими приложениями?

Очевидно, что самое лучшее решение – полностью от них избавиться. Спросите себя, действительно ли данная функциональность еще востребована. Довольно часто мы продолжаем эксплуатировать старые системы ради нескольких функций, которые невозможно воспроизвести где-либо еще, и скрытая стоимость поддержки таких приложений неочевидна; на выведение устаревших компонентов из системы отведено недостаточно ресурсов.

Если это не вариант, мы должны использовать архитектуру, которую разработчики уже давно используют для кода, – *паттерн подавления* (strangler pattern) [3]. При помощи этого паттерна мы пытаемся отстраниться от устаревших приложений, раз за разом перемещая все больше и больше функциональности в наши новые приложения. Со временем все меньше функциональности будет оставаться в устаревших приложениях, и, наконец, настанет ситуация, когда стоимость поддержки приложения только ради оставшихся функций будет признана слишком большой, и от него решено будет отказаться.

Последний прием, который вы можете применить к устаревшим приложениям, – визуализация стоимости их поддержки. Здесь могут фигурировать следующие факторы:

- задержки, с которыми сталкиваются другие приложения из-за устаревшего приложения;
- дефекты, возникающие из-за устаревшего приложения;
- количество средств, затраченных на поддержку и исполнение устаревших приложений, а также
- стоимость того, что вы не можете осуществить из-за устаревшего приложения.

Чем больше экономический ущерб от использования устаревших приложений, тем больше ваши шансы на то, что со временем вы убедите организацию начать что-то с ними делать.

Я уже говорил, что каждое из приложений, созданное вами, в ближайшем будущем устареет. В свете возрастающей скорости развития ИТ это вселяет опасения: скоро мы придем к тому, что сделанное нами сегодня уже завтра никуда не годится. Отсюда резюме: от устаревших приложений лучше избавляться, заменяя их новыми устаревшими приложениями, созданными с правильным подходом. Больше не существует конечной архитектуры (и, по сути, никогда не было, как стало известно на сегодняшний день, – несмотря на то что говорили нам архитекторы в компаниях). Согласно данному подходу каждое приложе-

ние должно создаваться так, чтобы его можно было бы легко отстранять и минимизировать его зависимость от других приложений. Мы глубже рассмотрим эту тему в главе 10.

Управление вашим портфолио и контрольными точками

Портфолио ваших приложений постоянно развивается, и, чтобы оставаться на плаву в такой подвижной среде, нужно использовать правильные подходы к управлению им. Ранее управление давалось тяжело, на сегодняшний день оно еще сложнее. Стало больше вещей, о которых нужно заботиться; сама скорость поставки изменений возросла, и если не изменять подходы, то управление или будет замедлять поставку, или станет неоправданно дорогим.

Ниже приведены четыре контрольные точки управления для любой перемены:

- контрольная точка 1 (КТ1): здесь мы должны ответить на вопрос, достаточно ли хороша наша идея об изменениях и заслуживает ли она финансирования, необходимого для того, чтобы продвигать идею дальше и находить возможные решения;
- контрольная точка 2 (КТ2): здесь нужно ответить на вопрос, нашли ли мы возможное решение, достаточно хорошее для того, чтобы совершить первые эксперименты или выпустить первый релиз, подтверждающий нашу идею;
- контрольная точка 3 (КТ3): здесь мы ответим на вопрос, достаточно ли качественно реализованное нами решение, чтобы выпустить его для хотя бы небольшой аудитории в среде эксплуатации;
- контрольная точка 4 (КТ4): здесь мы ответим на вопросы, был ли эксперимент успешным и что делать дальше.

Контрольная точка 1 (КТ1)

В КТ1 мы по большому счету будем говорить о заинтересованных сторонах. Где-то на просторах организации появилась хорошая идея или была обнаружена проблема, требующая внимания. Перед тем как начать тратить средства, согласно первой контрольной точке мы проверяем, правильные ли проблемы мы изучаем, а возможности – имеют ли они вес для бизнеса, являются ли крайне важными и способны ли наши «исследовательские идеи» привести нас к новым областям в бизнесе. Эта точка – ворота, проходя через которые мы убеждаемся в том, что не начинаем делать слишком много вещей одновременно и сосредоточили свои усилия на наиболее многообещающих идеях.

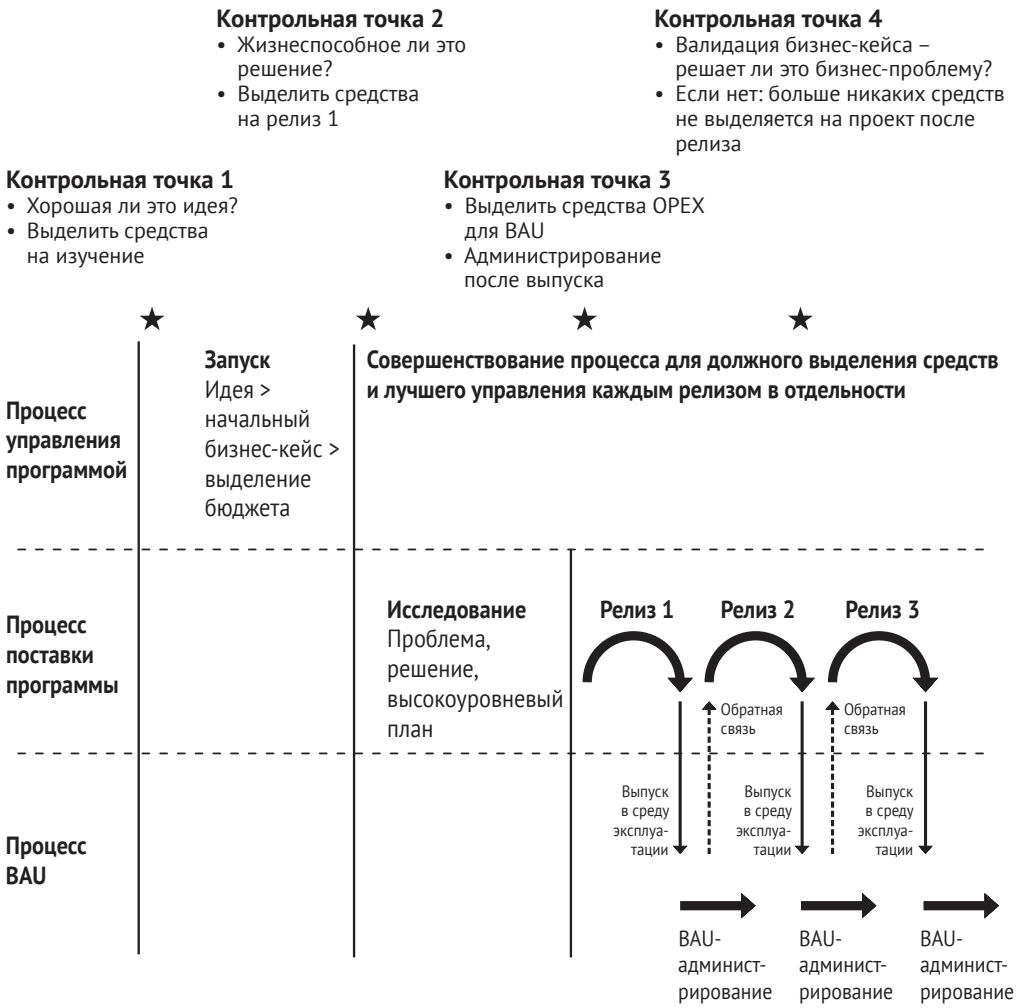


Рис. 2.3. Контрольные точки управления:
Agile-процесс управления с четырьмя контрольными точками

В положении между точками 1 и 2 организация изучает идею и бизнес, вместе с ИТ кооперируется для запуска *исследовательского* практикума, проведение которого может занять от пары часов до нескольких недель, в зависимости от масштаба проблемы. Вы можете проводить практикум в течение всей бизнес-трансформации или вовсе для небольших изменений. Цель исследования касается трех крайне важных аспектов: 1) все должны понимать проблему и идею; 2) мы изучаем, что может быть сделано при помощи ИТ, и 3) мы хотим понять, как могла бы выглядеть реализация, исходя из графика и команды. Эта исследовательская сессия крайне важна, так как она поможет людям в организации добиться наилучших результатов.

Контрольная точка 2 (КТ2)

Второй шаг после исследования – проверить, что мы обнаружили нечто, что стоит реализовать. На этой стадии мы должны убедиться, что у нас есть ресурсы для того, чтобы поддерживать реализацию со всех сторон: ИТ, представители бизнеса, специалисты по эксплуатации, безопасности и все, кого это касается. Это важная контрольная точка, на которой нужно собрать все архитектурные требования, так как потом их собирать будет уже труднее. Слишком часто бизнес-инициативы реализуются без должного осмысления архитектурных аспектов, что со временем ведет к увеличению технического долга.

Мне кажется, что каждая инициатива, которая поддерживается организацией даже с малыми финансовыми и человеческими ресурсами, принесет большую пользу организации в двух аспектах: она больше поможет бизнесу и улучшит ИТ-ландшафт. Это единственный разумный способ, который со временем сократит технический долг и поможет сладить с устаревшими приложениями. КТ2 – прекрасный период для того, чтобы убедиться, что на проекте запланировано улучшение ИТ-ландшафта и уменьшение технического долга, прежде чем дело дойдет до реализации. Это должно быть обязательной установкой, иначе пружинка постепенно вернется в исходное состояние. Довольно просто дать слабину и сказать себе, что «только один разок» мы быстренько реализуем временное решение. За много лет работы я понял, что нет ничего более постоянного, чем временные решения.

В промежутке между точками 2 и 3 цикл разработки продукта включает в себя дизайн, разработку и тестирование, осуществляемые в манере Agile. Я уверен, что Agile – единственная методология, которая нам нужна для того, чтобы двигаться дальше, но тогда в нашем процессе предоставления ИТ-услуг будут различные уровни мотивации и скорости поставки. Когда решение окрепнет, пройдя несколько итераций, оно станет кандидатом на релиз, а мы перейдем к КТ3.

Контрольная точка 3 (КТ3)

В контрольной точке 3 мы убедимся в том, что кандидат на релиз достиг необходимого уровня качества, чтобы можно было его выпустить в среду эксплуатации. Мы убедимся, что к идеям по архитектуре прислушались и технический долг был погашен, как было договорено, и мы не будем незаметно накапливать новый технический долг. (Иногда мы сознательно решаем взять на себя небольшой долг, чтобы пораньше что-либо протестировать, а потом закрыть долг в следующем релизе. Но все-таки подобное не должно происходить часто.) Эта контрольная точка часто ассоциируется с панелью управления изменениями, при помощи которой необходимо пересматривать и утверждать любые изменения, вносимые в среду эксплуатации. И конечно же, нам нужно добиться минимального жизнеспособного продукта, и вы можете обратиться

к предыдущей главе, чтобы припомнить общие принципы управления, которым можно следовать в контрольной точке 3.

В промежутке между контрольными точками 3 и 4 продукт находится в среде эксплуатации и используется. Если мы будем должным образом следовать Agile-процессу, то команда уже работает над реализацией следующего релиза, одновременно поддерживая версию, которая уже была выпущена в среду эксплуатации. Внешние или внутренние заинтересованные стороны пользуются продуктом, а мы получаем обратную связь напрямую из систем (мониторинг, аналитика и другие средства) или непосредственно от заинтересованных сторон при помощи опросов, анкет или каких-либо других каналов взаимодействия.

Контрольная точка 4 (КТ4)

Контрольную точку 4, по моему мнению, часто недооценивают. Это тот процесс, важность которого все понимают, но немногие готовы проявлять энтузиазм и решительность, чтобы воспользоваться его потенциалом. Эта точка нужна для того, чтобы убедиться в том, что наша идея и подход к ее решению верны. Так как проекты – явление временное по определению, к этому моменту команда начинает переформировываться, и некоторые сотрудники уходят в другие проекты. КТ4 начинает казаться формальным упражнением, которое люди не могут оценить сполна. Если наши команды долгое время существуют в неизменном составе, идея о том, что надо анализировать выводы предыдущих релизов, и мнение заинтересованных лиц становятся намного более важными. Эти проектные команды становятся ключевой аудиторией КТ4, хотя, конечно же, заинтересованные стороны в организации выступают другой аудиторией, которая должна видеть, что финансирование себя оправдало и что стоит инвестировать и далее.

КТ4 должна способствовать получению ценного опыта и дать возможность отпраздновать успех; негативные эффекты тут ни к чему. Если идея не сработала, то мы все равно получили полезную информацию о нашем продукте и нам стоит действовать как-то иначе в следующий раз. Вы можете использовать КТ4 вместе с оценкой внедрения, чтобы взглянуть на то, как поставлялся релиз, и усовершенствовать процесс и продукт в целом. Я лично предпочитаю проводить анализ (ревью) внедрения отдельно для усовершенствования продукта и для процесса поставки – как двух различных видов деятельности.

Используя модель управления и эти контрольные точки, вы сможете управлять процессом поставки с различной скоростью, а также справляться с быстрым развитием. Каждая контрольная точка позволяет вам оценивать прогресс и жизнеспособность инициативы; также вы сможете переносить инициативу на иную модель поставки, которая может реализовываться на другой скорости (быстрее или медленнее).

Первые шаги вашей организации

Чтобы закрепить знания, полученные в этой главе, попробуйте использовать в вашей организации следующие два упражнения. В этот раз они довольно тесно связаны друг с другом: первое касается анализа вашего портфолио приложений, а второе – определения минимального жизнеспособного кластера, для которого расширение возможностей принесет ценность.

Анализ портфолио приложений

Если вы похожи на большинство моих клиентов, то у вас ИТ-портфолио наверняка состоит из тысячи приложений. Если вы начнете изменять их все, то, вероятнее всего, не сможете увидеть прогресс и начнете думать, действительно ли выделенные средства оправдали себя в этих приложениях. Так что в первой главе упражнения были составлены для одного измерения, а область приложений – это уже второе, не менее важное измерение. Давайте посмотрим на то, как должны быть классифицированы ваши приложения.

Каждая организация обладает различной информацией о своих приложениях, но в целом можно проводить анализ по следующим четырем пунктам:

- Важность приложения: насколько важным для бизнеса является приложение? Насколько большое влияние проблемы с этим приложением окажут на пользовательское впечатление наших клиентов или работников? Насколько это приложение соответствует нормативным требованиям?
- Объем инвестиций для приложения: сколько средств мы выделим на это приложение в течение следующих трех лет? Сколько мы уже потратили на него? Сколько важных приложений будет связано с этим приложением в течение следующих нескольких лет?
- Предпочитаемая частота внесения изменений: если бизнес имеет возможность высказать предпочтения по частоте внесения изменений в приложение, то каков будет выбор: каждый час, неделю, месяц, год? Как часто мы вносили изменения в это приложение в течение последнего года?
- Технологический стек. Это важный аспект, так как некоторые технологии легче освоить, нежели другие. Вдобавок, как только у вас появится возможность ускорить поставку, например, приложения на Siebel, то и другие Siebel-приложения получится поставлять быстрее, так как инструменты, практики и методы – это то, что можно использовать многократно. Примите во внимание все аспекты приложения при осмыслении технологического стека: базу данных, сами данные, код, серверы приложений и инфраструктуру.

Для первых трех измерений вы можете использовать абсолютные значения (если они у вас есть) или относительные, которые будут представлять собой номинальную шкалу для оценки приложений. Для составления технологического стека вы можете группировать приложения в порядке приоритетности, исходя из вашего технического опыта в практиках DevOps по этим технологиям. Я рекомендую пользоваться таблицей с заголовками, подобной табл. 2.1. Пользуясь этой информацией, вы сможете упорядочить приложения согласно эвристическому алгоритму либо прибегая к сортировке вручную. Здесь важно не выработать идеально точное решение, а хотя бы добиться достоверных результатов.

Понятно, что мы не будем тратить много времени, сил и средств на приложения, которые редко изменяются, – приложения, которые не критичны для нашего бизнеса и на которые мы не собираемся выделять много средств в будущем. К сожалению, одного лишь упорядочивания приложений бывает недостаточно, так как ИТ-ландшафт организаций очень сложен и требует внедрения дополнительного уровня анализа для разрешения проблем с зависимостями в архитектуре приложений.

Таблица 2.1. Пример анализа приложений: подобная таблица поможет вам структурировать процесс анализа приложений

#	Приложение	Технология	Стратегическое приложение	Частота изменений	Объем, занимаемый приложением в портфолио
95	App A	Java, .NET, Oracle	4 – Критичное	9	4 – Очень большой

Определение минимального жизнеспособного кластера

Как обсуждалось ранее, минимальный жизнеспособный кластер – это ряд приложений, на которых вам нужно сосредоточиться таким образом, чтобы с их развитием ускорялась поставка всего кластера. Следуйте следующим шагам для определения минимального жизнеспособного кластера:

1. Выберите наиболее приоритетные приложения (в идеале основанные на анализе портфолио из предыдущего упражнения) для вашего начального набора приложений (состоящего лишь из одного приложения).
2. Обдумайте, какие другие приложения нужно изменять, чтобы осуществить изменение для выбранного набора приложений.

3. Разумно ограничьте список этих приложений (например, охватывающих 80 % обычного или планируемого объема изменений, вносимых в выбранное приложение).
4. Теперь у вас есть новый, увеличенный набор приложений, и вы можете повторять шаги 2 и 3, пока не добьетесь того, чтобы набор приложений превратился в минимальный жизнеспособный кластер.
5. Если кластер становится слишком большим, выберите другое начальное приложение или действуйте более радикально на этапе 3.

Когда вы справитесь с определением минимального жизнеспособного кластера, то сможете развивать процесс далее, реализуя практики DevOps, такие как автоматизация тестирования и применение облачного окружения, или формируя Agile-команду, которая будет поставлять изменения в этот кластер.

ГЛАВА 3

Готовые программные пакеты и поставщики программных продуктов

> «Если вы не знаете, что вам нужно, – говорит швейцар, – то вы получите много того, чего не хотите».
>
> *Чак Паланик*

Во многих организациях довольно нелестно отзываются о готовых программных пакетах – обычно это коммерческие продукты (COTS), в пользу которых был сделан выбор из-за поставляемой функциональности. Понятно, что это ПО, часто называемое устаревшим, не появилось в организацииниоткуда: кто-то решил приобрести программный пакет, для того чтобы решить бизнес-проблему. Существует множество весомых причин не изобретать колесо, а вместо этого пользоваться готовыми программными пакетами. К сожалению, множество программных пакетов сегодня не соответствуют тому, как должны выглядеть современные приложения. В этой главе мы обсудим критерии, которые должны рассмотреть при выборе программных пакетов, а также то, что вам стоит делать, чтобы улучшить имеющийся у вас программный пакет. Как всегда, я дам несколько упражнений в конце главы, чтобы вы могли лучше воспринять теоретическую часть. Давайте немного поговорим о том, что же представляют собой программные пакеты.

Изначально программные пакеты задумывались как средство поддержки стандартных процессов организации. Такие процессы не сильно отличаются

в разных организациях, и совсем не они определяют ваше отличие от конкурентов. И даже несмотря на то, что многие из этих программных пакетов сегодня поставляются в виде *ПО как сервис* (SaaS), в вашей организации наверняка работают устаревшие решения, на поддержку которых уходят ресурсы.

Проблема в том, что многие организации, применяющие программные пакеты, в итоге настолько существенно адаптировали продукт под свои задачи, что этот путь его совершенствования стал обходиться слишком дорого. Например, я наблюдал множество обновлений для Siebel, которые стоили миллионы долларов. Когда путь обновлений заводит слишком далеко, новая, улучшенная, более безопасная функциональность, которая поставляется с более свежими версиями пакета, часто оказывается недоступной для организации на протяжении нескольких лет. Кроме прочего, серьезная адаптация пакета под требования бизнеса со временем приведет к тому, что каждое следующее изменение будет обходиться все дороже и дороже и технический долг будет расти соответственно.

Еще если мы признаем, что концепция конечной архитектуры больше не применима, станет понятно, что программные пакеты должны проектироваться таким образом, чтобы они могли соответствовать жизненному циклу, который подразумевает естественное отстранение компонентов новой, отличной системы, чтобы можно было свободно отделять их от остальной архитектуры. Каждый компонент архитектуры должен представлять собой нечто временное, чтобы вы не были привязаны к какому-то одному поставщику. Рынок ИТ так быстро развивается, что выбор в пользу беспроигрышного варианта может завтра стать жизненной необходимостью. Иметь дело с пакетом, поставщик которого больше не присутствует на рынке или перестал считать этот продукт стратегически важным для себя, – это кошмар, ведь исправления дефектов и другого рода поддержки в таком случае крайне тяжело добиться. Я часто встречался с такими ситуациями, и мне приходилось заниматься обратным проектированием вместе с командой, для того чтобы решить проблемы, а это стоило немалых денег и времени.

Архитектура приложения – ключевой фактор, определяющий быстроту, с которой вы сможете поставлять продукт; и вместе с тем именно этот фактор тяжелее и дольше всего изменять. Вы должны осознавать, что, выдвинув решение внедрить новый программный пакет в организации, следует учитывать его влияние на архитектуру и не руководствоваться лишь желанием получить определенную функциональность.

Многие процессы ИТ-поставки сегодня больше напоминают сборку конструктора LEGO, чем лепку из пластилина. Чтобы вы могли такое осуществить, ваши приложения должны быть модулярными, пользоваться открытой архитектурой и хорошими практиками проектирования, – тогда вы сможете добиться наибольшей выгоды. В противном случае все это станет похоже на конструктор LEGO с различными механизмами соединения, и в один прекрасный момент вы придете к тому, что для поддержания всей конструкции нужен клей. Это подрывает саму идею использования конструктора, и изменения в такую структуру будет вносить крайне тяжело.

Как выбрать подходящий продукт для вашей организации

Касательно выбора ИТ-продукта для выполнения какой-либо бизнес-функции высказывалось множество мнений. Например, какую из CRM-систем стоит использовать: SalesForce, Siebel или Microsoft? Рассматривать лишь функциональность сегодня уже недостаточно: да, от нас зависит выбор того или иного продукта, но организация с наименьшей вероятностью станет использовать его в первозданном виде. Архитектура приложения, в состав которой войдет этот продукт, продолжит развиваться, что, скорее всего, потребует модификации продукта.

Принципы построения архитектур и проектирования сегодня играют намного большую роль, нежели ранее, благодаря постоянному развитию архитектуры. Это позволяет взглянуть по-другому на выбор продукта. И конечно же, выбор зависит от контекста для каждой компании и каждой бизнес-области. Фреймворк, который вы выберете, должен соответствовать всем рассмотренным факторам. И хотя это решение каждый будет принимать по-своему, я тем не менее представлю фреймворк выбора технологий (TDF), который поможет вам более широко взглянуть на ряд технологий, прежде чем сделать выбор.

В моем фреймворке TDF есть три стороны, которым необходимо давать оценку: 1) функциональность; 2) архитектура и 3) принципы проектирования.

Функциональность

Очень часто предоставляемая функциональность играет определяющую роль при выборе программного пакета. Чем больше функциональность соответствует требованиям процесса, который вы хотите обслужить, тем лучше выбор. Чтобы вы могли определить, подходит ли программный пакет или вам стоит создать свое собственное решение (надеюсь, не с нуля, а при посредстве открытых библиотек и модулей), вам необходимо внимательно взглянуть на свою организацию. Два фактора могут значительно повлиять на ваше решение: гибкость, с которой вы хотите обслуживать процесс, и ваши способности в проектировании. Если ваши процессы не очень гибкие и у вас оригинальный процесс, то используемый программный продукт, вероятнее всего, потребует серьезных изменений. Если у вас в штате нет людей с сильными навыками проектирования, да и ваши партнеры ничего не могут предложить в этом плане, то здесь наверняка подойдет использование программного пакета. Нужно четко видеть разницу между гибким процессом, созданным с низким навыком проектирования (случай для пакета), и самостоятельно созданным процессом, созданным с хорошим навыком проектирования (случай для оригинального решения).

Если вы останавливаетесь на использовании программного пакета, вам нужно составить список нужной функциональности в виде требований или *пользовательских историй*, а также необходимо оценить пакеты-кандидаты. В идеальном случае стоило бы привлечь к оценке представителей бизнеса,

чтобы они подтвердили, что в этой функциональности действительно есть необходимость. Основная идея состоит в том, что пакет поставляет готовую широкую функциональность, и для того чтобы заполучить ее, вам не придется тратить много сил на установку данного пакета в некотором окружении. Если трудности возникают, это сигнал, что надо что-то менять.

Зрелость (завершенность) архитектуры

Зрелость архитектуры приложения – весьма важный фактор для процесса текущей поддержки вашего приложения, так как хорошо спроектированное приложение облегчит задачу обслуживания и поддержки самого приложения. Если вы создаете приложение самостоятельно, то вам придется размышлять над архитектурными проблемами, например над масштабируемостью и мониторингом, и чем лучше при этом ваш навык проектирования, тем больше у вас возможностей самостоятельного построения этих архитектурных аспектов. Так или иначе, вы можете выбрать пакетное решение, которое будет делать это за вас.

Ниже представлены четыре критерия, которыми вы можете пользоваться для оценки зрелости архитектуры:

1. Автоматическое масштабирование: когда ваше приложение достигает успехов и начинает использоваться все чаще, нужно масштабировать функции, подверженные нагрузке. Архитектура должна разумно поддерживать гибкое масштабирование различных частей приложения (например, вы масштабируете не все приложение, а только ряд необходимых функций).
2. Самовосстанавливаемость: когда что-то идет не так, архитектура приложения должна быть способна распознавать такие ситуации и принимать контрмеры. Это может подразумевать традиционный перезапуск серверов/приложений, удаление очередей сообщений или запуск новой версии приложения/сервера.
3. Мониторинг: вам нужно понимать, что происходит с вашим приложением. Какие элементы используются? Какие части приложения несут ценность для вашего бизнеса? Чтобы это можно было узнать, архитектура приложения должна позволять вам наблюдать за многими его аспектами и предоставлять данные для вашего программного решения по мониторингу.
4. Способность к переменам: необходимо понимать, во что обойдутся попытки подстроить систему под текущие требования. Насколько данная архитектура модульна? Если в ней будет присутствовать множество общих компонентов, это помешает вам вносить независимые изменения и, вероятно, увеличит объем работы из-за наличия зависимостей в общих модулях. Архитектура приложения должна быть модульной по своей природе, чтобы у вас была возможность заменять и обновлять компоненты без необходимости изменения всей системы. Прямая и обратная совместимости также важны для обеспечения гибкости обновлений.

Принципы проектирования

Важность принципов проектирования растет вместе с вашей убежденностью в том, что приложению суждено расширяться; а это, в свою очередь, часто определяется степенью стратегической важности приложения для процессов взаимодействия с вашим заказчиком. Следование хорошим принципам при построении приложения позволит вам быстро обновлять его и масштабировать процесс поставки, чтобы можно было поддерживать растущие объемы вносимых изменений. Чем опытнее ваш ИТ-отдел, тем больше люди будут способны применять эти принципы и паттерны. Если такого навыка у ваших сотрудников нет, то вы сосредоточитесь на встроенных особенностях архитектуры. Вот несколько моментов, на которые стоит обратить внимание:

- Управление исходным кодом: весь код и конфигурации должны быть извлекаемыми. Вам важно использовать инструменты для управления конфигурациями в масштабах корпорации, для того чтобы обслуживать зависимости между системами. Следовательно, нужна возможность получать полную конфигурацию приложения и быстро ее восстанавливать. Встроенные или проприетарные (собственные) решения обычно не позволяют вам интегрироваться с другими приложениями, тем самым лишая ваши приложения возможности иметь некое единое состояние в пределах ваших корпоративных систем. При необходимости вы должны восстанавливать приложение до исходного состояния при помощи внешней системы управления исходным кодом, что осуществимо только в случае, если вы можете экспортировать конфигурацию из приложения в *систему управления конфигурациями программного обеспечения* (SCM). Это также значит, что конфигурации не должны быть применимы лишь к одному из готовых коммерческих продуктов. Легкость, с которой можно осуществлять экспорт и импорт конфигураций, покажет вам, насколько хорошо такой подход будет интегрироваться в жизненном цикле поставки вашего продукта. Извлекаемые данные должны представлять собой текст, чтобы SCM-системы имели возможность сравнивать различные версии, анализировать разницу и поддерживать слияние веток.
- Автоматизация при помощи API: приложение должно создаваться с учетом дальнейшей автоматизации, и в нем нужно оставлять средства (например, *программный интерфейс приложения*, API), которые позволят автоматизировать жизненный цикл поставки. Они включают в себя статический анализ кода, юнит-тесты, компиляцию и сборку. Ни одно из этих занятий не должно подразумевать использование графического пользовательского интерфейса. То же самое касается развертывания и конфигурирования приложения в целевом окружении; эти процессы развертывания и конфигурирования не должны подразумевать участия человека. В результате успешной автоматизации время сборки и развер-

тывания сократится (например, это будет занимать не более нескольких часов или, еще лучше, не более нескольких минут).

- Модульность приложения: это свойство сокращает время сборки и развертывания, обеспечивая выпуск в среду эксплуатации малыми партиями, а малый объем работы в среднем сокращает операционные издержки на внесение изменений. Это уменьшает вероятность того, что разработчики будут одновременно работать над одним и тем же кодом. Это, в свою очередь, сократит риски возникновения сложных случаев слияния веток.
- Облачные возможности: в первую очередь мы не должны строить монолит, поэтому нужно иметь возможность масштабировать необходимые компоненты, не затрагивая при этом приложение в целом. Существуют гибкие планы, которые могут поддерживать использование облачных решений. Механизмы встроены в систему таким образом, чтобы обеспечить мониторинг на низком уровне.

Для того чтобы помочь вам выбрать продукт, наиболее подходящий для ваших требований, оцените каждый из рассматриваемых продуктов по четырем критериям, упомянутым выше: функциональности, архитектуре, способности к проектированию и навыкам ИТ-отдела. Ниже представлен пример оценочной таблицы (табл. 3.1), которой вы можете пользоваться в качестве начальной точки для развертывания процесса оценки.

Таблица 3.1. Пример оценочной таблицы: новые приложения должны оцениваться по четырем критериям, а не только по функциональности

		Продукт А	Продукт Б	Продукт В
Функциональность	Функ. область 1			
	Функ. область 2			
	Функ. область 3			
Архитектура	Автоматическая масштабируемость			
	Самовосстанавливаемость			
	Мониторинг			
	Изменяемость			
Способности к проектированию	Исходный код			
	API			
	Модульность			
	Облачные технологии			
Навыки ИТ-отдела				

Что же тогда делать с существующими устаревшими приложениями

Рекомендации из предыдущего раздела прекрасно подходят для создания новых приложений, но вам все еще придется иметь дело с существующими

устаревшими приложениями. Позднее я расскажу, как развивать архитектуру ваших приложений, когда все ее элементы обслуживаются вами. Сейчас же мы обсудим, что можно сделать, когда приложения не полностью обслуживаются вами.

У вас наверняка есть ряд приложений, над которыми вы работаете, и некоторые из них поддерживаются поставщиками программных пакетов, которые занимаются созданием ПО, или внесением в ваше ПО специфичных изменений, или тем и другим. Но тем не менее если вы взглянете на приложение, то вряд ли сможете сказать, что оно следует современным принципам построения архитектуры и проектирования, как я говорил ранее. Весьма вероятно, что вы не захотите вкладываться в полную замену этих систем, поэтому вам придется поискать другие методы обхождения с ними. Существует четыре основных принципа, которым я рекомендую следовать при работе с такими поставщиками: 1) пользуйтесь вашими системными интеграторами; 2) пользуйтесь ассоциациями пользователей; 3) сокращайте пользование функциональностью приложения и/или 4) подталкивайте поставщика улучшать продукт.

Понятно, что количество поставщиков не соответствует количеству приложений. Существуют мультимиллионные организации, с которыми вам придется взаимодействовать иначе, нежели с мелкими поставщиками, поддерживающими одно приложение.

Пользуйтесь системными интеграторами

На протяжении всей своей карьеры я работал с поставщиками ПО или системными интеграторами и много раз дивился тому, столько упускается возможностей, между тем как стоило бы наладить эффективное взаимодействие, и не только ради выполнения срочного задания. Если вы работаете с крупным системным интегратором (SI) для поддержки и разработки приложения, то весьма вероятно, что SI способен работать со многими аспектами приложения. В то время как вы работаете с некоторым поставщиком ПО, SI, исходя из потребностей ряда его клиентов, может больше воздействовать на поставщика. Чем лучше у вас взаимопонимание с SI в части разработки, тем легче вам сообща влиять на поставщика. Тесная работа с поставщиками ради улучшения их архитектуры – только одно из многих преимуществ, к которому ведет изменение ваших отношений с SI.

Пользуйтесь ассоциациями пользователей

Среди большинства популярных приложений часто формируются ассоциации, которые могут стать действенным каналом для получения обратной связи от поставщика. Иногда они организуются самим поставщиком, а иногда независимо от него. В любом случае будет полезно найти единомышленников, которые хотят улучшить архитектуру приложения в соответствии с современными практиками. И обращение целой группы клиентов к поставщику может оказаться более действенным подходом. Несколько лет назад я работал

с некоторым ПО, которое не было способно предоставлять отчеты на основе *стори-поинтов*, а позволяло лишь пользоваться отслеживанием рабочих часов. Поставщик продукта всегда говорил моему клиенту, моим коллегам и мне, что наш запрос в его практике единичный и потому не имеет для него высокого приоритета. Мы смогли чего-то добиться только тогда, когда связались с некоторыми другими организациями, которые, как ни удивительно, тоже делали такой запрос и получали такой же ответ! Поставщик явно чего-то недоговаривал. Как только мы смогли объединить усилия, поставщик начал серьезно воспринимать наши запросы и исправил проблему.

Я рекомендую вам искать такие пользовательские группы, чтобы найти потенциальных союзников, а также способы решения проблем, к которым вы впоследствии сможете прибегнуть. Сейчас люди со всего мира находят решения, для того чтобы использовать практики DevOps в тех приложениях, в которых формально не получается реализовывать эти практики. И к счастью, сторонники DevOps обычно весьма рады поделиться этой информацией.

Отгораживайтесь от этих приложений и вкладывайтесь в нечто новое

Как уже было сказано в предыдущей главе, когда приложения не изменяются и вы уже считаете, что пора от них отказаться, можно воспользоваться чем-то, подобным паттерну подавления, медленно отстраняясь от использования приложения. Сокращайте инвестиции в это приложение и вкладывайтесь в построение новой функциональности – той, которая будет лучше сочетаться с вашей архитектурой. Будьте честны и сообщайте вашему поставщику ПО о том, что вы делаете это, поскольку он не предоставляет необходимых вам возможностей, но вы готовы пересмотреть свое решение, если такие возможности появятся. Для поставщика это серьезный мотив вкладываться в создание улучшенной инфраструктуры (ведь эта функциональность, возможно, не существует как раз потому, что о ней никто ранее не просил). Последовательно объясните, почему неразвивающаяся архитектура и инструменты не помогают вам двигаться дальше и для соответствия вашим современным требованиям об архитектуре необходимо вносить изменения в архитектуру приложения. Если поставщик решит, что эти возможности не нужно реализовывать в приложении, вряд ли вам стоит с ним сотрудничать: полезнее вложить ваши средства куда-либо еще, чтобы ваша архитектура вышла на новый уровень.

Подталкивайте поставщика к действиям

Я всегда предпочитаю пряник, а не кнут; в идеале стоит искать возможности достижения взаимовыгодных договоренностей. Совершенствование архитектуры и проектирования принесет выгоду для вас; этим вы можете воспользоваться, чтобы подтолкнуть поставщика к изменениям в работе. Чтобы еще

больше заинтересовать его, можно показать, как изменения сделают приложение более привлекательным для вашей организации и насколько больше лицензий необходимо будет приобрести со временем. Это заметно поспособствует тому, чтобы поставщики улучшали свою архитектуру. И конечно же, вы можете рассказать другим, какое замечательное у вас приложение, и тем самым привести еще больше клиентов – выигрывают все.

Как я говорил в начале главы, многие организации не очень высокого мнения о программных пакетах. Тем более удивительно, что организации пассивно подходят к работе с пакетами и редко подталкивают поставщиков к исправлению ситуации. Я верю, что поставщики будут рады пойти навстречу, если больше организаций будут задавать правильные вопросы. В конце концов, зачем поставщику вкладываться в архитектуру, подходящую для методик DevOps и Agile, если каждый из клиентов запрашивает лишь новую функциональность, а не улучшение архитектуры? Если компании плотнее будут работать с поставщиками и обсуждать методы, при помощи которых они хотят управлять программным пакетом, и то, насколько важны для них в этом процессе практики DevOps, поставщики начнут обращать на это больше внимания.

Будьте креативными

Если ничего из этого не сработает и вы смелы и любознательны, то можете проигнорировать руководства ваших поставщиков и попытаться приспособить техники DevOps самостоятельно – даже к программным продуктам, которые не так легко поддаются этим техникам. Вот как вы можете начать:

- Найдите и обслуживайте исходный код. Правда, проще это сказать, чем сделать. Вам, возможно, придется извлекать конфигурации из базы данных или из файловой системы, чтобы получить текстовую версию кода.
- Ищите способы управления этим кодом при помощи общих конфигураций и инструментов для слияния кода, вместо того чтобы пользоваться оригинальными системами, которые мог порекомендовать поставщик. Стоит также рассмотреть синтаксис кода, чтобы узнать, присутствуют ли в нем части кода, которые представляют собой несоответствующие метаданные, которые вы можете пропустить во время процесса слияния. Нечто подобное в Siebel, например, позволило моей команде сэкономить сотни часов.
- Попытайтесь найти в приложениях API или сигнальщики, с помощью которых можно автоматизировать шаги процесса, в противном случае требующие ручного вмешательства (даже если изначально предполагалось, что они будут использоваться для других целей).

В моей команде мы использовали эти техники в таких приложениях, как Siebel, SAP, Salesforce и Pega[1].

[1] Я составил более детальное описание этих техник в блоге *InfoQ* [1].

Я надеюсь, что техники, описанные выше, помогут вам проложить свой путь и стать частью экосистемы, в которой ИТ – настоящий двигатель. Последний элемент экосистемы, о котором я хочу вам рассказать, – роль системного интегратора. Эта тема очень мне близка, и о ней я поговорю в следующей главе.

Первые шаги вашей организации

Определите основные принципы работы новых приложений

Составьте оценочную таблицу для вашей организации, взяв за основу табл. 3.1. Принимайте следующее решение по продукту (называйте его историческим, если хотите), пользуясь этой оценочной таблицей, и посмотрите, появились ли отличия в процессе. Проследите, приносит ли использование оценочной таблицы, сосредоточенной на архитектуре, заметный результат. Я бы порекомендовал приглашать заинтересованных лиц из организации, чтобы проводить практикумы с обсуждением результатов и предстоящих шагов к переменам по мере вашего продвижения вперед.

Создавайте поддерживающую экосистему для улучшения вашей архитектуры

Итак, в вашей организации, как и во многих других, уже имеются программные пакеты. В предыдущей главе мы проанализировали портфолио приложений, которым вы можете пользоваться, для того чтобы определять, какие из программных пакетов будут стратегически важными для вашей организации.

1. На основе проведенного анализа портфолио приложений (или на основе чего-либо еще) определите небольшой ряд стратегических приложений (например, как первый минимальный жизнеспособный кластер), чтобы сформировать стратегию для создания поддерживающей их экосистемы.
2. Теперь выберите стратегические программные пакеты и пройдитесь по оценочной таблице, описанной в этой главе. По большому счету, вы можете не ориентироваться на функциональные аспекты, так как они используются при выборе между пакетом и собственным решением. Тем не менее вы могли бы пройтись по всей таблице, чтобы проверить, является ли ваше решение правильным. Учитывая, что вы будете делать эту оценку уже постфактум, вы узнаете, насколько подходящим оказался пакет, основываясь на количестве адаптаций решения, выполненных вашей организацией.
3. Продумайте стратегию работы со слабыми местами в программных пакетах. Каким образом работать с поставщиком, чтобы улучшить ваши возможности? Будете ли вы работать с ними напрямую? При-

влечете ли вы системного интегратора или будете искать пользовательские группы?

4. Для получения результатов нужно время. Определите реалистичную частоту проведения инспекции улучшенной экосистемы, чтобы понять, помогает ли она улучшить приложения, над которыми вы работаете. Вы можете использовать принципы измерения технического долга из предыдущей главы в качестве отправной точки, если у вас не найдется иных средств для оценки улучшений в пакетных приложениях.

ГЛАВА 4

Поиски подходящего партнера

> Партнерство подобно бракосочетанию. Все усилия и лучшие намерения в мире не принесут должных результатов, если вы в первую очередь выбрали не того партнера.
>
> *Сесилия Грант,*
> *«Рождество, которое пошло совершенно не по плану»*

В большинстве крупных организаций работа не ведется в одиночку. Где-то в вашей организации для мелких или крупных задач в рамках ИТ-отдела привлекаются сторонние работники, или, по крайней мере, SI помогает вам поставлять продукт, который необходим вам, чтобы вести бизнес. Нужно правильно управлять подобными отношениями, дабы быть уверенным, что вы получаете надлежащую интеллектуальную собственность и работаете с опытным партнером.

В индустрии ведется множество разговоров о том, как важна культура, и о том, что Agile и DevOps стали, по большому счету, культурными движениями. Вы услышите множество историй о том, как улучшить культуру в организации, когда будете посещать конференции или вести блоги. Я полностью согласен с тем, что культура организации крайне важна для достижения успеха, но мне любопытно, почему сейчас ведется множество дискуссий о том, как выстроить культуры SI и организаций, с которыми они работают. На сегодняшний день большинство взаимоотношений компаний и SI основаны на сделках и на управлении отношениями с поставщиками. Такие слова, как *партнер*, *парт-*

нерство и *сотрудничество*, нередко используются, но на деле между «партнерствующими» сторонами по ряду причин нередки разногласия.

В этой главе я хочу помочь вам улучшить ситуацию, представив свой опыт работы и с SI, и в качестве клиента SI. Существуют методы улучшения взаимоотношений, которые помогут сделать их более значимыми для обеих сторон. Есть и распространенные ошибки, которых стоит избегать. Каждая из сторон прежде всего хочет наладить взаимовыгодные отношения – по крайней мере, так было в моей практике. Зачастую расширению организационной культуры мешают отсутствие контекста и ограниченный опыт.

Как добиться выгодных стратегических партнерских отношений с системным интегратором

Многие организации, которые следуют пути Agile и DevOps, решают, что наилучший способ успешного перехода к этим практикам – изначально полагаться на штатные возможности, так как при этом сохраняется больше возможностей управлять сотрудниками и средой, в которой они работают (учитывая их зарплату, цели, мотивацию, принятую внутреннюю политику), чем в случае работы с SI.

До тех пор пока вы не захотите перевести все процессы в обратно штат, вы, вероятнее всего, будете работать с SI-партнерами. К счастью, в работе с партнерами есть множество преимуществ. Подходящий партнер способен поделиться с вами опытом работы с другими компаниями, у него наработаны более богатые связи с вашими поставщиками, а кроме того, он предоставит для вас среду, привлекающую таланты, – среду, которую вы сами создать не смогли бы. ИТ сегодня являются основой всякого бизнеса, но не каждая компания способна стать ИТ-компанией. Наличие стратегических партнеров позволит вам пользоваться преимуществами каждой из сторон – иметь достаточно прав на интеллектуальную собственность и представления о том, как создаются и работают ваши системы, в то же время позволяя вашему стратегическому партнеру оперировать большим объемом ИТ-задач. Не бойтесь и будьте готовы передавать ИТ-задачи при необходимости, чтобы поддерживать баланс – и в целом оставаться успешными.

Мир технологий меняется крайне быстро; это означает, что нам постоянно необходимо узнавать новые технологии. Если у вас будут хорошие отношения с партнером, то вы сможете вместе взяться за изучение новых технологий и обеспечить обучение сотрудников вашего партнера, а в то же время получить выгоду от успехов партнера в освоении этой новой технологии. Я всякий раз умиляюсь, наблюдая, как две компании работают вместе над поиском взаимовыгодных решений. Важно проявлять внимание к интересам вашего партнера.

Работая над некоторыми проектами, я входил в состав смешанных команд, в которых опыт моих людей в технологиях использовался вместе с глубокими познаниями в бизнесе сотрудников клиента. Эти команды клиента могли поддерживать и улучшать наше решение даже спустя долгое время после нашего ухода; именно в этом и проявляется успех! Мы не только создали улучшенную систему, но еще и обогатили организацию, подтянув навыки сотрудников в новых для них методах работы. И, как обсуждалось ранее в главе о портфолио приложений, у вас смогут быть приложения, к работе с которыми вы не хотите привлекать штатных сотрудников и с которыми данный подход не сработает.

В контексте ваших основных и вспомогательных приложений вам нужно использовать технологии и опыт предыдущих проектов у SI, а также познания в бизнесе и области – право на интеллектуальную собственность в вашей ИТ-организации. Стоит избегать партнеров, которые не ориентируются на предпочтительные методы работы в вашей организации и процессы и культура которых для вас непрозрачны, а следовательно, нельзя быть уверенным, что они соответствуют вашим. В противном случае знания о ваших системах останутся только у отдельных сотрудников таких партнеров, и все изменения будут происходить в т. н. *режиме «черного ящика»*. В итоге, когда что-то пойдет не так, вы не сразу это поймете. Один из способов решения проблемы выбора поставщиков – сужение их ряда до небольшого количества стратегических партнеров, ради которых вы готовы потратить усилия на то, чтобы сделать партнерство успешным. Чем меньше партнеров, тем меньше будет зависимостей при попытках приблизиться к их культуре. Культурное сближение в принципах работы, способах мотивации, ценностях, а также требуемом опыте должно стать основным критерием при выборе SI, помимо стоимости.

Важно контролировать свой путь в ИТ

Вашей организации необходимо понимать, как работает ИТ, и обладать достаточными возможностями, навыками и интеллектуальной собственностью, чтобы самостоятельно определять свою судьбу. Как говорилось ранее, ИТ сейчас находится в самом сердце бизнеса; минимальное понимание того, как это работает, важно для того, чтобы с вашей помощью ИТ поддерживали бизнес сегодня, завтра и послезавтра. Но что же это означает – контролировать свою судьбу в ИТ? В то время как существуют некоторые тенденции, снимающие головную боль с вашего ИТ-отдела (cloud, SaaS, COTS), на самом деле нет способа стопроцентной отслеживаемости рисков, связанных с ИТ.

Вам также придется думать об инструментах и процессах, которые принесут с собой ваши партнеры. Прекрасно, когда поставщик привносит дополнительные инструменты, методы и т. п., но если вы не сможете продолжать использовать эти инструменты и методы после смены поставщика, то окажетесь один на один с крупной проблемой, если они успели стать крайне важными для всей ИТ-поставки. Если отношения с поставщиком непрозрачны и вы не вполне по-

нимаете, как он работает, то вам придется брать на себя все риски таких взаимоотношений: вы ведь связаны с ним прочнее, чем, возможно, этого желаете.

К счастью, существует тенденция движения в сторону методов и стандартов, которые облегчают взаимодействие, несмотря на барьеры, присутствующие в компании. Хорошие примеры – такие Agile-методологии, как SAFe и LeSS. Очень вероятно, что вы будете приспосабливать ваши собственные методы, основанные на влиянии многих фреймворков. Когда этот метод становится обязательным условием для работы организации, вам легче все контролировать. И все равно стоит убеждаться в том, что ваши методы верны, и не бояться обратной связи от партнеров. Ваши партнеры, безусловно, должны привносить свой опыт, что поможет вам усовершенствовать свои методы.

Следование стандартам – неплохая практика на стороне проектирования. Многие организации либо не оказывают влияния на разработку решений своими партнерами, либо не видят этот процесс. Такие практики, как модульное тестирование, статический анализ кода и автоматизированное развертывание, лежат в основе всего. Но, так или иначе, многие организации не имеют представления о том, используют ли эти практики их партнеры, а если да, то в какой мере. Поддержание правильной структуры и взаимодействия может помочь вашим партнерам начать пользоваться этими практиками, но от вас зависит, сможете ли вы начать видеть, какие практики применяются для ваших проектов.

Довольно практичный подход в такой ситуации – наличие в организации стандартов проектирования, которых будет придерживаться каждая из ваших команд, будь то штатные сотрудники, поставщик или множество поставщиков. Вместе с этими стандартами у вас появится общий язык, на котором вы сможете говорить с партнерами, когда будете описывать свое видение предоставляемых вами ИТ-услуг (например, ваше определение непрерывной интеграции). К счастью, в этом направлении уже было проделано много работы, и не нужно начинать все сначала. Для вдохновения можете почитать популярные книги по разработке: «*Непрерывное развертывание ПО. Автоматизация процессов сборки, тестирования и внедрения новых версий программ*» Джеза Хамбла и Дэвида Фарли, «*Программист-прагматик. Путь от подмастерья к мастеру*» Эндрю Ханта и Дэвида Томаса, а также «*Release it! Проектирование и дизайн ПО для тех, кому не все равно*» Майкла Т. Найгарда.

Смена парадигмы «Разработка – администрирование – внедрение»

Ранее контракты с системными интеграторами работали в макиавеллиевском стиле: компания создавала невыносимую среду, в которой никто не выходил победителем. Одна из моделей, которая невольно испытывала подобные последствия, – это модель «*Разработка – администрирование – внедрение*» (DOT). Не знаю, насколько вы знакомы с этим понятием, поэтому давайте я коротко расскажу, что имеется в виду. Контракты DOT работают на основе

представления о том, что существуют три отдельные стадии проекта: стадия разработки, когда продукт создается, стадия администрирования, когда продукт поддерживается другой стороной, и стадия внедрения, когда продукт возвращается в штатное обслуживание.

Многие организации пользуются услугами двух различных поставщиков для осуществления разработки и администрирования или обдумывают передачу стадии администрирования другому поставщику, одновременно сотрудничая с партнером по разработке. Есть несколько причин считать эту модель неверной. Во-первых, если ваш партнер отвечает лишь за поставку, то по понятным причинам соображения об администрировании не будут иметь для него большого веса. В конце концов, не ему же предстоит осуществлять администрирование! Сторона, которая будет заниматься этой фазой, точно так же будет отстаивать лишь свои интересы, все чаще освещая проблемы ближе к моменту передачи продукта. Здесь нет злого умысла, просто каждый сосредоточивается на своей зоне ответственности – в зависимости от круга задач, очерченного контрактом.

Вторая проблема в том, что многие DOT-проекты ведутся по принципу «черного ящика», когда клиентская организация вовлечена лишь в качестве заинтересованной стороны и до тех пор, пока не подоспеет стадия внедрения в штат, не имеет представления о том, как поддерживать и обслуживать систему. Это приводит к возникновению проблем не только на стадии внедрения, но еще и при урегулировании несоответствий в момент перехода со стадии разработки к администрированию. Вы можете изменить эту модель вовлеченности в переход от одной стадии к другой, так чтобы она соответствовала ряду характеристик. Убедитесь в том, что обеспечиваете длительное существование команды на протяжении нескольких стадий, осуществляемых вашими партнерами, чтобы сотрудники, которые будут администрировать решение, были уже вовлечены в разработку.

На протяжении всего жизненного цикла проекта внедряйте своих людей в команду, чтобы вы могли расширять свои представления о данном решении и о том, чего стоит его создавать и поддерживать. В идеальном случае желательно найти пару таких представителей от вашей организации и от вашего партнера (например, в качестве проектного менеджера, лида команды поставки, системного архитектора), чтобы можно было разделить обязанности. Такая модель предусматривает правильное обхождение с недостатками DOT-модели, и с ней вы все-таки сможете пользоваться общей концепцией DOT-проектов и комбинировать ее с принципами DevOps. Лучшие из моих проектов пользовались такой моделью, и результаты ее применения функционировали еще довольно долгое время, не требуя моего вмешательства.

Культурное взаимодействие в партнерстве

Как я упоминал ранее, мне приходилось бывать по обе стороны партнерства с системным интегратором (SI), предоставляя услуги клиенту, а также вы-

ступая в роли ответственного за совершенствование навыков сотрудничества, когда я работал с SI. Довольно легко винить SI в каких-то неполадках, в том, что не применяются все практики DevOps и Agile, а также в том, что он не готов экспериментировать, чтобы улучшить дела.

Действительность такова, что каждый человек и каждая организация, оглядываясь на свой контекст, делает то, что считает правильным. Никто не пытается достичь заведомо плохих результатов. К сожалению, иногда взаимоотношения выстраиваются на недоверии: я не доверяю тебе, поэтому у меня будет человек, который будет следить за тем, что ты делаешь. Поставщик затем выделяет отдельную должность, и обе стороны увеличивают количество процессов и документации, чтобы прикрыть свои тылы. Больше и больше процессов, ролей и прочих сложностей – пока не появится несколько отдельных уровней, отдаляющих настоящую работу от сотрудников обеих организаций, взаимодействующих друг с другом. Будет еще хуже, если эта деятельность не привносит никакой ценности, а только усугубляет взаимное недоверие партнеров.

Представьте, что вы доверились вашему SI как наилучшему сотруднику в команде. Какими процессами и документами можно пренебречь и как это повлияет на стоимость и быстроту поставки? Несмотря на эти потенциальные преимущества, выстраиванию такого эффективного взаимодействия уделяется слишком мало внимания. Как можно создать культуру сотрудничества, которая будет мотивировать всех партнеров двигаться вперед, преследуя идею выстроить рабочие методы Agile и DevOps, и как нам это сделать, если у нас уже есть долгосрочные контракты?

Во-первых, я думаю, что важно понимать вашего партнера. Как в хорошем браке, необходимо знать, что работает в отношениях с партнером, а что – нет. И в слово «партнер» я вкладываю особый смысл. Если вы работаете с поставщиком над неким сторонним проектом и отношения между вами преимущественно формальные, то вам нет повода о чем-либо беспокоиться. Но если вы работаете с одной и той же компанией в течение многих лет над какими-то основными системами, формальности уже не настолько значимы. Вам необходимо выстроить партнерские отношения и выработать общую DevOps-культуру.

В действительности вы можете понять, как SI определяет свою успешность, и каждая из сторон не боится показать, что рассчитывает получить от этого взаимодействия. Один из весомых факторов – карьерные мотивы, и мне, в свою очередь, повезло, так как множество моих клиентов с пониманием относились к тому, что я обсуждал карьерный рост своих сотрудников и объяснял, почему мне нужно менять их текущие роли на проектах. При взгляде со стороны компании кажется, что выгоднее иметь одного человека на одной и той же роли в течение многих лет, но меня это категорически не устраивало, поскольку моим сотрудникам было интересно как-то развиваться. И конечно же, у другой стороны также могут быть и свои интересы, поэтому надо задуматься, если SI использует непрозрачную тактику – вам нужно, чтобы взаимоотношения были как можно более понятными! Вы, конечно, можете приобрести услуги,

как кота в мешке, и пользоваться ими лишь при помощи интерфейса. В таком случае вам все равно, как много сотрудников работает над проектом и что они делают; вы просто платите за предоставление услуги. Это дает SI право на независимость – и вы, возможно, встретитесь с аспектами ваших ИТ-стратегий, которые могут работать в манере XaaS.

В случае с другими проектами, которые подразумевают работу с вашими основными системами и сотрудниками из вашей или сторонней организации, вам нужна прозрачность. Поставщик, который привносит свои инструменты и методы, в принципе, увеличивает стоимость вашего перехода к другим моделям. Вам нужно иметь ваши собственные методы и инструменты, и каждый SI может поделиться своим опытом, чтобы помочь вам улучшить их. Вам не нужен «черный ящик». К счастью, общие фреймворки в индустрии, такие как SAFe или Scrum, действительно помогают выстроить базовый подход к совместной работе организаций, подразумевающий малые траты времени на подготовительные действия. Обдумывая партнерство, вы должны помнить о том, что нельзя совершенно исключить риски. Часто я встречал клиентов, которые просто говорят: «ну, это уже ваша проблема», – когда SI начинает описывать возможное развитие ситуации. На самом деле если проект провалится, то клиент пострадает в первую очередь. Закрывать глаза и уши и перекладывать проблему на плечи SI – непростительная слабость. Вспомните о катастрофе Australian census, когда партнер по поставке вынес на публику тонны негатива [1], или Healthcare.gov в США, поставщики которой обвиняли друг друга в проблемах [2]. Даже если поставщики были и вправду виноваты, репутация той и другой организации понесла потери, а с учетом того, что это были публичные сервисы, появление множества негативных отзывов в прессе было вполне предсказуемо.

Контракты с партнерами

Используя Agile, мы хотим добиться гибкости и прозрачности процессов. Но насколько хорошо вы ради этого проработали свои контракты? В таком случае в контракте уже нельзя упоминать все ту же фиксированную оплату и фиксированные результаты, которые используются тогда, когда каждое изменение должно подвергнуться тщательной проверке. Контракты часто составляются на основе некоторых представлений, но если их не придерживаться, это может привести к проблемам. Упоминание времени и материалов в Agile-контрактах может навлечь проблемы, так как это не будет способствовать тому, чтобы решения принимались согласно предыдущим результатам работы – такое может сработать, только если ваш партнер имеет большой опыт в Agile и ваши взаимоотношения зрелые и выстроены на доверии.

Составление Agile-контрактов потребует от вас, чтобы вы уделяли больше времени отслеживанию результатов работы и вовлечению в регулирование объема работы. По моему опыту, лучшие Agile-контракты строились на наличии фиксированных возможностей, достижении некоторого результата

и гибком регулировании объема работы (поставка определенной функциональности, подробности реализации которой определяются по мере развития проекта).

Существуют способы составления Agile-контрактов, которые будут выгодны каждой из сторон, поэтому давайте познакомимся с некоторыми основами типичного Agile-проекта. Agile помогает командам по завершении каждого из спринтов поставлять код, готовый к среде эксплуатации, и процесс поставки делится в этих целях на четыре фазы:

1) изучение объема работы и предстоящая работа по уточнению деталей (*definition of ready*);
2) поставка пользовательских историй по завершении спринта/итерации (*definition of done*);
3) подготовка к релизу/совершенствование и передача в среду эксплуатации (*definition of done-done*);
4) пострелизная поддержка / гарантийное обслуживание (*definition of done-done-done*).

По сути, контракт нужно составлять с учетом этих четырех фаз. Назначайте размер стоимости работы, основываясь на артефактах поставки или на фазах, когда ваш партнер получает оплату только после поставки (например, проектных документов) или по завершении некоторой стадии проекта (например, стадии проектирования или разработки): такие контракты отражают суть пользовательских историй в виде очков для пользовательских историй. Каждая из историй проходит стадии, описанные выше, и с этим должна быть связана оплата. Стоит, конечно, выделять определенную долю выплат за историю, которая достигает стадии definition of ready, а также других таких стадий. Ниже приведен хороший пример такого деления.

- У нас есть три сотни очков для пользовательских историй, которые нужно поставить в течение трех итераций, а также один релиз в среду эксплуатации: итоговая стоимость – $1000.
- График выплат – 10/40/30/20 % (первая выплата – на старте проекта, вторая – по завершении итераций работы над историями, третья – по выпуске их в среду эксплуатации, последняя выплата – после некоторого периода гарантийного обслуживания).
- Подписание контракта: 10 % = $100.
- Итерация 1 (50 очков завершено): 50/300 × 0,4 × 1000 = $66.
- Итерация 2 (100 очков завершено: 100/300 × 0,4 × 1000 = $133.
- Итерация 3 (150 очков завершено): 150/300 × 0,4 × 1000 = $201.
- Отладка и выпуск в эксплуатацию: 30 % = $300.
- Гарантийные обязательства: 20 % = $200.

Пользуясь такой моделью для составления контракта, мы получим модель, которая сможет связать поставку объема работы и размер вознаграждения

поставщика. По моему опыту, эта модель выступает связующим звеном между обеспечением гибкости и оплатой лишь необходимого объема работы. Есть еще некоторый элемент, который вам бы захотелось включить в ваш контракт: опытного представителя, который способен принимать своевременные решения, определенные принципы управления и рабочую среду, которая сопутствует методам Agile-поставки (физическое рабочее пространство, ИТ, инфраструктура и т. д.). Зрелые организации упоминают затраты времени и материалов, поскольку используют свою надежную методологию для обслуживания качества и объема результатов. Менее же опытные организации прекрасно управляются с контрактами, составленными на основе фаз, упомянутых выше.

Еще одним аспектом составления контрактов выступает мотивация. Представим ситуацию: у вас налажены многолетние рабочие отношения с SI, но, все еще обслуживая устаревшие приложения, вы не подумывали о том, чтобы применять DevOps-практики. Теперь вы хотите исправить положение дел. Вы договариваетесь о схеме совместного инвестирования и вскоре соглашаетесь о единой схеме работы с вашими приложениями. Несколькими месяцами позднее вы видите первые положительные результаты на демонстрациях непрерывной интеграции и автоматизации тестирования. Ваш SI подходит к вам на одной из обычных встреч и предлагает обсудить с вами контракт по причине того, что изменилась среднедневная рабочая ставка у его команды. Что вы ожидаете увидеть в будущем? Не то ли, что среднедневная рабочая ставка будет уменьшаться в результате автоматизации? Я имею в виду, она станет менее оплачиваемой, не так ли?

Давайте присмотримся к этому вместе. Среднедневная ставка – это средняя ставка, рассчитываемая исходя из того, что менее квалифицированная работа стоит дешевле, а работа, подразумевающая наличие определенных навыков, более дорогая и оплачивается соответственно. Соотношение этих двух типов оплаты и определяет среднедневную ставку. Когда дело доходит до автоматизации, что мы автоматизируем в первую очередь? Конечно же, работу, которая требует меньше навыков! Автоматизация же сама по себе требует больших навыков. Это означает, что доля высококвалифицированной работы увеличивается, а вместе с ней и среднедневная ставка. Но позвольте… не означает ли это, что вся работа дорожает? Нет. Так как мы заменили некоторую работу автоматизированными задачами, то в перспективе все должно, наоборот, удешевиться. Если вы все еще оцениваете ваших SI по средней дневной ставке, то вам нужно переосмыслить свою оценку. То, что для нас важно, – это не средняя дневная ставка, а общая стоимость.

Диаграмма на рис. 4.1 может помочь вам визуализировать этот противоречивый вывод. В то время как среднедневная ставка за выполнение одного и того же объема работы увеличивается (относительная доля высококвалифицированной работы растет), общая стоимость уменьшается (так как в итоге получается, что работы, выполняемой вручную, меньше). Довольно много

организаций пользуются показателем среднедневной ставки для оценки своих партнеров и тем самым способствуют дальнейшему применению низкоквалифицированной ручной работы вместо автоматизации. Автоматизация – это действительно взаимовыгодный сценарий, так как системный интегратор получает возможность сократить риски, связанные с задачами, выполняемыми вручную, но в то же время увеличивается уровень среднедневной ставки. В итоге клиент тоже извлекает выгоду из того, что сокращаются риски и общий показатель стоимости.

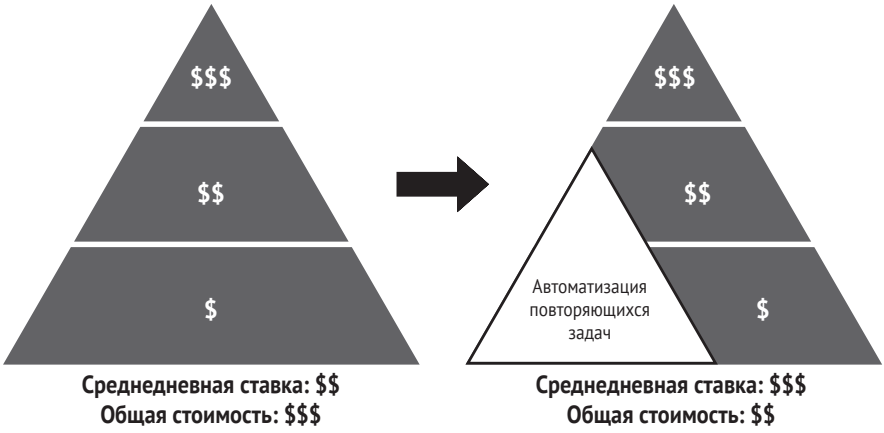


Рис. 4.1. Общая стоимость и среднедневная ставка: работа по автоматизации сокращает общую стоимость, но увеличивает среднедневную ставку

Партнерство со стороны SI

Также я хочу взглянуть на взаимоотношения поставщика сервиса и компании с позиции системного интегратора. Об этой стороне мало кто говорит, но учитывая то, что это обычно *моя* сторона, я бы хотел рассказать, что мы об этом думаем и какие препятствия видим.

Влияние культуры DevOps начало менять отношения в сторону более открытых. Даже в запросах на процессы поставок я наблюдаю все большую открытость к обсуждению объема работ и подхода к поставке. Я работал с клиентами из правительства и прямо во время процесса имел возможность помочь им сменить запрос на нечто, более соответствующее тому, чего они добивались, когда переняли составление Agile-контрактов, наподобие тех, что я упоминал ранее. Изначально они хотели осуществлять оплату по завершении каждой из поставок, что не очень соответствует принципам Agile-поставки. Вместе мы обычно приходили к некоему решению, которое работало для обеих сторон и позволяло убедиться в том, что все получают запланированный результат.

Часто считают, что именно системные интеграторы обязаны вести переговоры с отделами по снабжению, которые имеют мало общего с поставкой современных технологий. Контракты составляются с такой эффективностью, что не остается места для экспериментов, а там, где эксперименты возможны, единственным приемлемым результатом будет положительный. Анализируя взаимоотношения с вашими SI, подумайте о способах повысить доверие, стать более открытыми и более эффективно преследовать свои цели. В конце этой главы я привожу небольшой тест для определения того, насколько тесно смыкаются ваши культуры.

Оценка партнера

В завершение хотелось бы поговорить об оценке партнера. Понятно, что вас никак не устроит партнер, который за немалую плату предоставит организации неудовлетворительную услугу. Так как же обеспечить достойные взаимоотношения, при этом стремясь к открытой культуре? И к тому же сохранить некоторые механизмы влияния на ситуацию, если производительность начнет падать?

Возьмем для примера проект, которым занимается некоторый SI. Можно использовать оценочные таблички, которые учитывают несколько аспектов.

Один из них – поставка; в этой области вам важны качество и предсказуемость. Как много дефектов ускользает в эксплуатацию, насколько точны предположения о поставке и как отслеживаются финансы? Вам, возможно, также захочется использовать оценку заинтересованными сторонами качества предоставления ИТ-услуг в виде потенциального внутреннего показателя лояльности клиентов (NPS, который отображает долю людей, рекомендующих ваш сервис) от заинтересованных сторон из бизнеса. Время продолжительности цикла и объем работ – вот два показателя, которые вам стоит использовать для того, чтобы улучшить рабочие процессы в целом.

Второй аспект – это техническое мастерство. В первую очередь вам необходимо добиваться соответствия вашим методам проектирования (модульное тестирование, автоматизация тестирования, непрерывная интеграция, непрерывная поставка…). Если ваш партнер по поставке использует легкие пути, то тем самым он будет накапливать технический долг, и в какой-то момент вам придется за это расплачиваться. По моим наблюдениям, клиенты хорошо справляются с проверкой качества конечного продукта, но часто не уделяют время тому, чтобы научиться следовать практикам проектирования, которые предотвращают нарастание технического долга. Обеспечение четкого и понятного ряда ожиданий от методов проектирования и регулярная их проверка (например, при демонстрации автоматизации развертывания, статической проверке кода и т. п.) сокращают шансы на дальнейшее увеличение технического долга. Я проводил развернутые дискуссии с клиентами о стратегиях проектирования, когда представлялся случай для таких демонстраций.

Третий аспект – пропускная способность. Вы можете рассматривать этот аспект с учетом очков пользовательской истории или даже с учетом количества историй, выпускаемых за один релиз. Я полагаю тем не менее, что ваши релизы начнут отличаться по структуре, по мере того как ваши профессиональные возможности будут расти. Интересный момент, касающийся продолжительности цикла: если вы оптимизируете его скорость, вы зачастую будете добиваться качества и меньшей стоимости, не прикладывая больших усилий, как говорилось выше.

Также вам стоит оставлять некоторое количество оценочных таблиц для улучшений. Какими узкими местами вы занимаетесь в данный момент и как составляете планы, учитывающие ваши предположения? Здесь вы можете использовать стоимость предоставления одной услуги (например, стоимость развертывания или стоимость выпуска в среду эксплуатации), чтобы видеть специфичные улучшения ключевых сервисов своей организации.

И последнее, но не менее важное: учитывайте интересы вашего партнера. Продвижение в карьере, предсказуемость и две другие области интересов, которые вам наверняка захочется обсудить, – доход и рентабельность; является ли ваше партнерство взаимовыгодным? Я рекомендую учитывать этот аспект в оценочных табличках, в которых вы отслеживаете две или три задачи, являющиеся приоритетными для вашего заказчика, и регулярно оценивать все показатели вместе.

Первые шаги вашей организации

Каждому свое – поиски подходящих партнеров

Вся эта глава была посвящена поискам правильного партнера, который будет соответствовать вашим запросам и культуре. Но, вероятнее всего, для поддержки различных элементов вашего портфолио понадобятся различные партнеры. Если вы не проигнорировали задание по составлению портфолио из главы 2, то это упражнение будет легче выполнить. Нам пригодятся три типа приложений:

- ключевые приложения. Эти приложения развиваются невероятно быстро и обычно непосредственно доступны клиентам, а также являются фактором, определяющим восприятие вашей компании на рынке;
- «рабочие лошадки». На этих приложениях держатся все основные процессы в вашей организации, например управление взаимоотношениями с клиентами, выставление счетов и процессы каналов поставок. На них часто ссылаются как на системы для корпорации или как на устаревшие системы. Они могут не быть непосредственно доступными клиенту, но компания получает значительную ценность от применения этих приложений и продолжает вносить в них изменения, чтобы поддерживать развивающиеся необходимости бизнеса;

- истинно устаревшие приложения. Они довольно стабильны и не требуют внесения многих изменений. В общем и целом они используются для поддержки ваших основных стабильных процессов или побочных аспектов вашего бизнеса.

Основываясь на этих классификациях, пересмотрите свою стратегию работы с партнерами, чтобы понимать, нужно ли вам менять партнера или стоит развивать тесное сотрудничество с уже проверенным. Для обслуживания первых двух категорий приложений понадобится развивать сотрудничество со стратегическими партнерами. Для работы с устаревшими приложениями стоит искать партнера, который предложит выгодные условия по стоимости работы и которому вы будете платить за поддержание жизнеспособности системы. Продумайте особые методы мотивирования ваших стратегических партнеров. Ваших партнеров по «рабочим лошадкам» стоит оценивать по степени эффективности услуг, предоставляемых для обслуживания приложения. В случае же обслуживания ключевых приложений партнер должен уметь проявлять гибкость и быть готовым вносить нововведения вместе с вами.

Проводите практикум со стратегическими партнерами по вашим «рабочим лошадкам»

Организации тратят львиную долю средств на своих «рабочих лошадок». И для этого есть весомые причины, так как эти приложения выступают фундаментом всего бизнеса. Для выполнения этого упражнения вам желательно пригласить на практикум своих стратегических партнеров, которые поддерживают ваши приложения («рабочих лошадок», а также ключевые приложения). Вы можете делать это вместе со всеми партнерами, но это будет труднее осуществить, либо вы можете провести практикум с каждым из них в отдельности. Важно убедить их в том, что текущую структуру контракта можно уточнять и изменять, а на протяжении всего практикума стоит быть открытыми к обсуждению.

Ниже представлена структура самого практикума:

- Объясните вашему партнеру, что для вас важно с точки зрения приоритетов бизнеса и ИТ.
- Обсудите, как вы можете измерять успешность с учетом ваших приоритетов.
- Попросите партнера сформулировать, что важно для него во взаимоотношениях с вами и что ему требуется, чтобы эти взаимоотношения рассматривались как успешные.
- Вместе подумайте, как вы можете соблюсти ваши интересы.
- Активно поразмышляйте над тем, какие цели стоит преследовать, чтобы достичь взаимовыгодных договоренностей между вашими организациями.

Ключевым моментом в этом практикуме должны стать обоюдная открытость и стремление к взаимовыгодному сотрудничеству. Моя практика показывает, что для достижения такого результата может потребоваться несколько заходов – не расстраивайтесь, если все проблемы не решатся прямо во время практикума. Как и все, о чем мы говорим, – это тоже итеративный процесс, и, возможно, вы поймете, что еще не нашли подходящего партнера и нужно внести некоторые изменения в вашу экосистему.

Проводите самостоятельную оценку культуры партнерства

Небольшой тест для оценки вашей DevOps-культуры взаимоотношений с системным интегратором:

- Используете ли вы среднедневную ставку в качестве показателя для измерения продуктивности, соотношения ценности и стоимости и т. д.?
  +1, если вы сказали «нет».
- Есть ли у вас разработанный механизм, который позволит вашему SI наряду с вами получить выгоду от того, что он улучшит свои техники и подходы при помощи автоматизации или других практик?
  +1, если вы сказали «да». Не стоит ожидать, что SI будет вкладывать все силы в новые практики, если для него это не принесет выгоды. С чисто этической позиции он, конечно, не прав, но давайте будем честны. У всех нас есть экономические цели, и если не обсуждать их с вашим SI, то и заинтересованность в деле будет наблюдаться только с вашей стороны.
- Есть ли у вас и вашего SI возможность маневра, которая обеспечит поле для экспериментов, суть которых понятна вам обоим?
  +1, если вы сказали «да». Вам необходимо знать, как много времени SI тратит на апробацию новых инструментов или практик. Если сейчас SI просто получает достаточно бюджета, который обеспечит выполнение задачи ровно таким образом, который ожидается, то начните просить о выделении бюджета на инновации и управляйтесь с ними вместе с SI.
- Готовы ли вы принять или хотя бы гипотетически допустить риск неудачи при проведении экспериментов?
  +1, если вы сказали «да». Если вы выделяете бюджет на инновации, сможете ли вы остаться хладнокровным, когда ваш SI сообщит вам о том, что идея по совершенствованию не сработала? Или вы готовы принимать только успешные эксперименты? Думаю, для вас очевиден ответ, который соответствует культуре DevOps.
- Знаете ли вы о том, что ваш SI подразумевает под успехом?
  +1, если вы сказали «да». Понимать цели вашего SI важно; это касается не только финансовой выгоды, но еще и сотрудников, которые работают

на SI. Мотивы карьерного роста и другие личные интересы должны приветствоваться, чтобы взаимоотношения были успешными.

- Договариваетесь ли вы с вашим SI без посредников?
  +1, если вы сказали «да». Если в этом процессе участвует еще одна сторона, например ваша команда по снабжению или внешний поставщик, то весьма вероятно, что диалог будет вестись по схеме «испорченного телефона». И нет гарантии, что ваши команды имеют представления о наилучших практиках DevOps для работы с поставщиками. Обсуждаете ли вы потенциальное наличие помех для составления контракта с вашим SI?

0–2 балла: у вас весьма формальные взаимоотношения с вашим SI, и вам стоит подумать о том, чтобы познакомиться с ним получше и улучшить ваши взаимоотошения. *3–4 балла*: вы неплохо справляетесь, но у вас есть куда расти, поэтому вам стоило бы провести практикум со своим партнером и обратить внимание на новые стороны развития. *5–6 баллов*: у вас по-настоящему партнерские отношения, которые вас будут вдохновлять в течение всего процесса трансформации. Прекрасная работа!

Заключение части A

Итак, в части A мы поговорили о создании мотивирующей экосистемы для обеспечения хороших результатов ИТ-поставки. Материал этой части имеет стратегическую ценность: изменения в описанные области будут вноситься только с течением времени, и для этого потребуются усилия всей вашей организации. Не пытайтесь изменить все в одночасье – я бы предпочел, чтобы вы усвоили изложенные мною идеи, проделали некоторые упражнения, а затем составили план на следующий год, следующий квартал, следующий месяц. Начните с проработки терминов, которые для вас новы. Это тоже не дело одного дня – вникайте в терминологию постепенно. В следующей части речь пойдет о ваших сотрудниках. Собственно, работу делают люди, поэтому, чтобы достичь успеха, вам необходимо оказывать им поддержку.

ЧАСТЬ Б

Люди и организация процессов

Во второй части этой книги я хотел бы обратить внимание на центральную составляющую всего, что мы делаем: на людей. Без них не обходится ни одна организация. Главная проблема, связанная с человеческим фактором, – пристрастие людей к шаблонным способам действий. Если у вас есть старые приложения, одни люди будут на них жаловаться, а другие – говорить, что от добра добра не ищут. Так как же все-таки переменить отношение сотрудников к устоявшимся принципам работы? И как структурировать организацию, чтобы такой образ действий привел к созданию хороших продуктов? Во второй части книги мы рассмотрим различные аспекты устоявшегося мышления, которое вам необходимо улучшить или сменить.

Берри Шварц на конференции TED talk в марте 2014 года высказал прекрасную мысль о том, что создаваемые нами системы сами по себе создадут работников, которые наилучшим образом будут с ними управляться [1]. Если мы создадим структуру, для которой требуется примитивная, постоянно повторяющаяся деятельность, то получим таких же примитивных сотрудников, которые только и могут совершать рутинные действия, как рабочие на сборочном конвейере. Я считаю, что нам нужно создать ИТ-организации, в которых реализуются процессы мышления, подобно тому как Дэн Пинк раскрывал в своем исследовании то, что мотивирует людей: автономия, совершенствование и цель [2]. Людям нужна автономия для того, чтобы делать собственный выбор, нужно совершенствование навыков, чтобы четко понимать, что они делают, и нужна цель, чтобы знать, зачем они это делают. Со временем все люди, работающие в организации, начнут делать все более значимую и стоящую работу, а при таком раскладе все будут в выигрыше. На протяжении второй части книги мы будем вновь и вновь возвращаться к этой идее.

Мы обсудим, как предоставить вашим сотрудникам правильное окружение для принятия решений, какая структура команды вам нужна, для того чтобы задействовать автономию и целеустремленность, как преодолеть инертное

мышление, которое загубило множество проектов по автоматизации, и как лучше управлять вашими людьми в постиндустриальном мире.

Кто-то однажды сказал мне, что жить было бы намного легче, если бы нам не приходилось иметь дело с другими людьми. Смею предположить, что это не вариант, если, конечно, вы не можете приобрести во владение целый остров, поэтому давайте поразмыслим, как облегчить общение с сотрудниками, создавая для них адекватные условия работы.

ГЛАВА 5

Контекст во главе угла

> Мудрость – это разум, который учитывает контекст.
>
> *Рахил Фарук*

Одной из целей Lean-практик является избавление от балласта. Но, как мы уже обсуждали, компоненты и результаты традиционного производства более осязаемы, чем в ИТ, что означает, что на производстве выявить нечто лишнее получится легче, чем в мире ИТ. На производстве мы можем физически увидеть отходы, посмотреть на продукты производства, которые хранятся на складе, посчитать количество производящихся в данный момент экземпляров, а также легко заметим, где машина простаивает. В ИТ мы имеем дело с нематериальными результатами, что заметно осложняет наблюдение за процессом. И если сотрудник в ИТ не занят своей непосредственной работой, мы не всегда это заметим. Вспомним закон Паркинсона: «Работа занимает столько времени, сколько на нее отведено».

Наилучший способ вовлечения всех ИТ-работников в такую деятельность, которая исключает ненужные шаги или простои, – дать им понять контекст их работы. Это позволит им проявлять инициативу, вместо того чтобы уныло ожидать чьей-то отмашки. Это позволит им принимать правильные решения, а также избегать пустой траты сил. Все это очень хорошо соотносится с факторами мотивации Дэна Пинка: контекст позволит вашим сотрудникам действовать самостоятельно и понимать цель своей работы.

В нашей организации, придерживающейся устоявшихся принципов, мы верили в производственный подход к поставке, согласно которому передача продукта от машины к машине не будет препятствовать получению результатов в том случае, если они являются частью отлаженного процесса. Машинам или рабочим на сборочном конвейере не нужно знать, почему надо вкрутить винт

в металлическую плиту именно в этом месте. Здесь вообще нет альтернативных решений для размещения винта, поэтому контекст не так важен.

В ИТ среда имеет значение, так как успешно реализовать специфичную функциональность получится только в случае, если ты понимаешь контекст, в котором функция будет использоваться. Я сам был разработчиком и знаю: когда ты ограничен техническим проектом без описания среды, то не можешь полноценно решать возникающие проблемы. Но если ты понимаешь контекст, то для решения всех проблем придумаешь хоть что-то, пускай и не вполне идеальное.

Когда компании запускают проект, они тратят много времени на разработку идеи, выстраивание бизнес-целей и во многих случаях – на их доработку, перед тем как непосредственно начать работу по проекту. Уже создан большой объем контекста, но во многих организациях этот контекст передается проектной команде лишь в виде документации. Более того, проектная команда имеет свойство меняться, и когда на проект приходят новые тестировщики, они смогут изучить контекст только по устаревшей документации и по «испорченному телефону».

К тому же важно заметить, что вместе с передачей будет теряться часть контекста. К сожалению, никакое количество комментариев и документации не в состоянии описать весь контекст. Я испытал это на себе, еще будучи специалистом по функциональному тестированию, когда при заведении дефекта мне присылали код, неспособный решить проблему, с которой я столкнулся.

Но все же важно передавать контекст информации, даже если часть его будет потеряна в процессе. Люди принимают решения, основываясь на контексте и мотивации. Редко какой вовлеченный в проект сотрудник будет злонамеренно принимать неудачные решения. Обычно это происходит из-за того, что имеющийся контекст не был учтен. Так как же мы можем создать распределенный контекст, который поможет принимать правильные решения?

Опыт участия в успешных Agile-проектах/программах/инициативах подсказывает мне, что фаза исследования может решить эту проблему. Общее ознакомление с проектом, или функциональностью, или каким бы то ни было элементом со временем окупится. Я буду использовать пример ограниченного Agile-проекта, который работает на протяжении нескольких релизов, но вы можете адаптировать его под любой размер.

Практика ознакомления может занять от нескольких часов до множества недель, поэтому ее лучше разделить на несколько фаз:

- понимание бизнес-проблемы;
- нахождение жизнеспособного решения проблемы;
- планирование поставки и подготовка.

Понимание бизнес-проблемы

В первую очередь при ознакомлении с проектом нужно добиться того, чтобы команда и заинтересованные стороны говорили об одном и том же. Чтобы

этого достичь, вам нужно собрать вместе как можно больше людей, которые будут участвовать в создании продукта. Если команда поставки невелика, она должна присутствовать на обсуждении в полном составе. Я вел некоторые Agile-инициативы, для которых требовалось наличие нескольких десятков людей в команде поставки; в таких случаях я приглашаю представителей каждой команды и специализации (например, разработчиков, бизнес-аналитиков, тестировщиков, специалистов из команды обслуживания мейнфреймов, из Java-команды). Вы запросто можете начать сессию с тридцатью присутствующими в зале. (SAFe даже предлагает механизмы для приглашения большего количества людей при помощи церемонии планирования программного инкремента.) Цель начальной стадии ознакомления с проектом – понимание того, какое решение будет искать вся команда.

Учитывая, что почти всем инициативам в организации нужно иметь финансируемую бизнес-проблему, то с бизнес-проблемы неплохо бы и начать. Спонсор представляет проблему команде, так чтобы каждый знал, что ожидается и как будет измеряться успех. Так как каждое из открытий для команды связано со специфичным контекстом, я предлагаю вам некоторые общие практики, которые помогут определить этот контекст и получить целостное понимание бизнес-проблемы:

- текущий и ожидаемый опыт заказчика / заинтересованного лица. Это поможет объяснить, какое влияние инициатива будет оказывать на наиболее важных людей – ваших клиентов. Для этого могут использоваться проектирование процессов, схемы воздействия, пути клиента или составление цепочки ценностей (если вам незнакомы эти техники, обратитесь к Google, он много об этом знает);
- персонажи. Некоторые команды могут более живо представить себе клиента при помощи создания персонажей, которых можно использовать во всей фазе поставки для написания пользовательских историй. Иные из моих команд невероятно активно пользовались персонажами – вплоть до того, что представляли, будто персонаж является частью команды;
- краткая презентация. Вам нужно создать некоторую краткую презентацию или формулировку ценностей, которые позволят команде сосредоточиться на том, чего вы желаете достичь. Они могут изменяться со временем, но останутся точкой опоры, к которой вы постоянно будете возвращаться;
- списки элементов, которые будут или не будут включены в проект. Некоторые элементы важны в Agile, так как они присутствуют в традиционной поставке. Не забывайте четко определять, что должно входить в проект, а что (еще важнее!) не должно входить;
- риски и проблемы. Обговорите возможные риски, постарайтесь выявить и устранить проблемы;
- зависимости. Продумайте изначальные зависимости внутри вашей организации и вне ее;

- обхождение с заинтересованными сторонами. Заинтересованные стороны являются ключевыми составляющими любого проекта. Вы можете создать идеальное решение, но если заинтересованные стороны останутся недовольными, считайте, что это провал. В связи с этим я люблю устраивать тренинги по переговорам с заинтересованными сторонами. В ходе этих тренингов команда определяет, на кого из них повлияет проект и как нужно с ними общаться. В Agile существуют более действенные каналы взаимодействия, чем в традиционных проектах. Подобные тренинги позволяют наладить коммуникацию, ответив на ряд важных вопросов:
  – Кто будет приглашен на презентации и совещания по планированию?
  – Кто будет принимать участие в сессиях по доработке бэклога?
  – Кому мы будем передавать записи демонстраций?
  Делайте обдуманный выбор, так как собрания могут превратиться в бардак, если в них будет принимать участие слишком большое количество заинтересованных сторон. В частности, на презентации некоторые могут отвлекать всеобщее внимание на «хорошие идеи», которые расширяют элементы, включенные в проект, или изменяют их;
- приоритизация и критерии успеха. Больше всего мне нравится на встречах по ознакомлению с проектом вести обсуждение о приоритизации и критериях успеха. В Agile все элементы проекта приоритизируются. Это позволяет избежать проблемы, при которой 90 % элементов являются обязательными к исполнению. Обсуждение четких критериев, по которым будут расставляться приоритеты, с представителями бизнеса и заинтересованными сторонами может даваться не так легко. Но именно на них вы будете опираться во время поставки, для того чтобы люди могли говорить об одном и том же и принимали более объективные решения. Очень часто проекты заходят в тупик из-за неполноценных требований от ключевых заинтересованных сторон, основанных на субъективных предпочтениях, а не на объективных результатах. Это же касается и критериев успеха: чем будет руководствоваться организация после выхода на рынок в оценке успешности проекта? Постарайтесь воздержаться от желания использовать подход «Узнаю́, когда вижу»[1]. Вам нужно найти хотя бы пару измеряемых признаков успеха, а само измерение и определение основных признаков должно стать частью проекта. К сожалению, сегодняшняя ИТ-реальность такова, что слишком малый ряд проектов проходит оценку успешности после выхода на рынок. Ситуация облегчается, если команды потом продолжают обслуживать проект – не просто существуют для выпуска проекта, а имеют долю собственности в продукте или сервисе.

[1] https://ru.wikipedia.org/wiki/%D0%A3%D0%B7%D0%BD%D0%B0%D1%8E,_%D0%BA%D0%BE%D0%B3%D0%B4%D0%B0_%D0%B2%D0%B8%D0%B6%D1%83. – *Прим. перев.*

Поиски жизнеспособного решения проблемы

Теперь, когда команда понимает контекст проекта, мы можем погрузиться в поиски решения проблемы. К тому же вы выполнили часть этой задачи при создании бизнес-кейса (в конце концов, вам ведь нужно было определить тех, кто будет участвовать в ознакомительных сессиях). Скорее всего, у вас уже есть хорошее представление о том, какие приложения будут затронуты, и мысленно вы продумали высокоуровневое решение, на котором будет основываться бизнес-кейс.

Эта стадия ознакомления может быть немного хаотичной, так как в ней есть много места для исследований. Вам хочется, чтобы все принимали участие в поисках наиболее подходящего решения проблемы. Для этого вы захотите совершить некоторое предварительное проектирование и составить техническую архитектуру решения. При необходимости вы будете погружаться в составляющие некоторых процессов или в специфичные технологии. Это прекрасный способ определения рисков и подтверждения представлений о том, что технология способна предоставить вам. Там, где еще присутствуют остаточные проблемы, вы можете выявить возможности, которые в дальнейшем захотите использовать на стадии поставки.

Вам придется найти для своей организации правильное соотношение между подробным документированием и совершенным отсутствием документации. При этом надо учитывать, что слишком большой объем документации потребует слишком много усилий и времени, в то время как ее отсутствие повлечет появление трудностей при реализации выбранного решения. Я рекомендую для начала составлять диаграммы на досках, в презентациях PowerPoint или в Visio. Высокоуровневая архитектурная схема и ее описание легко умещаются на доске и в памяти людей.

Совещания по процессам проектирования и технической архитектуре будут итеративно совершенствоваться по мере того, как вы будете их пересматривать, потому как решение для проекта со временем тоже будет меняться. Вы создали насыщенный контекст в первой части стадии ознакомления. Старайтесь периодически к нему возвращаться, чтобы убедиться, что вы учитываете рамки проекта и все, что было определено ранее, а важные детали остаются в фокусе. В некоторых случаях вы захотите преобразовать выводы, составленные в первой части, – это нормально. Проделав это несколько раз, вы сами удивитесь тому, как выводы первой части позволяют привести дискуссию к плодотворному результату. Небольшая подсказка: разместите выводы, сделанные на стадии ознакомления, на доске, чтобы их всегда было видно.

Вы также можете начать записывать планируемые функции решения, которые могут быть размещены в связанных эпиках (крупных кусках функционала). Этот список уже может быть исчерпывающим, или вы можете оставить себе возможность в дальнейшем определить дополнительные функции. Что вам нужно определить в любом случае, так это минимально жизнеспособный

продукт (англ. *minimum viable product*, сокр. MVP). Какой минимальный объем работ нужно покрыть, чтобы выйти на рынок? В дальнейшем это будет очень важно при планировании релиза. Но существует тенденция к раздуванию MVP, поэтому будьте избирательными при составлении объема работ, чтобы он действительно был минимальным. В случае крупных организаций со сложной архитектурой я придерживаюсь правила: MVP должен быть готов через три месяца. В вашем случае срок может отличаться.

Но подождите, разве вы не говорили, что Agile должен быть гибким? Почему мы определяем решение на стадии исследования, а не просто позволяем ему развиваться по мере итераций? По опыту работы в крупных организациях могу сказать, что вам пригодятся некоторые рамки. Я видел, как проекты затягивались на месяцы, когда после трех итераций заинтересованные стороны решали переместить функциональность из CRM-системы в ERP-систему. Структуру команды приходилось поменять, также это влияло на работу маркетинга, на стратегии тестирования и на многое другое. Универсальная идея гибкости – это не то, что способна реализовать инициатива. Думайте о стадии исследования как об эскизе к большой картине, который вы будете его дополнять деталями на этапе поставки. Ваша задача – не сформировать абстрактный образ, а нарисовать портрет прекрасной леди.

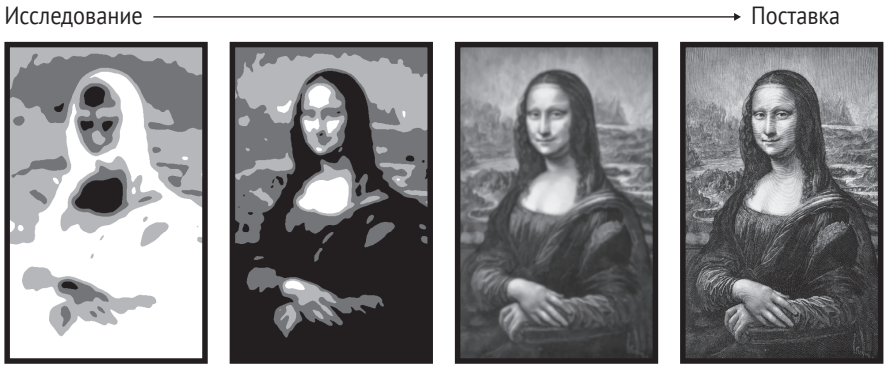


Рис. 5.1. Исследование и поставка.
Исследование – эскиз картины, поставка наполняет ее деталями

Планирование стадии поставки и подготовка к ней

После определения контекста и нахождения жизнеспособного решения завершающим этапом на стадии исследования будет подготовка к стадии поставки. Это очень важный этап, и он, скорее всего, займет некоторое время, так как подразумевает мобилизацию команд, настройку рабочего места, предварительную оценку работы и планирование графика релиза. В этот этап вовле-

чены не столько заинтересованные стороны из бизнес-сектора, сколько ИТ-команда поставки.

В зависимости от того, как ваша организация относится к Agile и DevOps, в первую очередь стоит выстроить привязку к терминологии, методологии и практикам, используемым в проекте. Лично я предпочитаю применять широко распространенные в индустрии фреймворки, такие как масштабируемый гибкий фреймворк (англ. *Scaled Agile Framework*, сокр. SAFe), поскольку он позволяет с самого начала пользоваться общепринятыми терминологией и практиками. И конечно же, вы можете применять свою собственную методологию или разработать новую. Небольшое предостережение: если вы решите заменить термин только потому, что вам не нравится, как он звучит (например, «скрам-мастер» на «координатор поставки», как я делал для клиента), то не сможете пользоваться поддержкой сообщества. В итоге вы не сможете найти такой термин в Google, и люди вне вашей компании не смогут понять, о чем речь. Еще один аспект, о котором стоит побеспокоиться, – общая структура иерархии работы. Я могу показать вам, что происходит, если у вас ее нет. Если вы используете:

- история > функциональность > задача;
- элемент бэклога (PBI) > задача > тема;
- история > тема > задача

в одной и той же организации, рано или поздно это приведет к недопониманию. Это уже будет гибкость ради гибкости, достигаемая ценой эффективности организации. В конечном счете начнется путаница в терминологии, поэтому просто выберите один набор терминов и придерживайтесь его. Не будьте пристрастны, это всего лишь слова; последовательность в работе намного важнее, чем личные предпочтения. Вы также можете проводить обучающие сессии для команды, владельца продукта и любых других ключевых ролей в период планирования этапа поставки.

Есть и другие важные вещи, на которые нужно обратить внимание перед началом работы, а именно:

- изначальная структура команды;
- технические практики (например, непрерывная интеграция или непрерывная поставка);
- идеальная структура релиза (ежемесячная, еженедельная, ежедневная…);
- как выглядит ваша техническая экосистема.

Техническая экосистема особенно важна в организации с большим количеством устаревших приложений.

Среды, как правило, ограничены, и вы должны координировать свои действия с другими подразделениями организации, поэтому понимание доступных ресурсов становится критически важным элементом успеха.

Также это подходящее время для того, чтобы провести начальную оценку объема работ по проекту. Для этого вы можете использовать покер планирования

или какую-либо другую Agile-практику для подготовки бэклога. Вам нужно будет стимулировать процесс, чтобы уже к началу первых итераций был определен объем работ, соответствующий представлениям команды о готовности: это поспособствует эффективности и успешности ее работы. В дальнейшем будет проводиться подготовка бэклога – исходя из моего опыта, потребуется время для того, чтобы скорость подготовки начала соответствовать скорости поставки. Я встречал много команд, у которых иссякал объем работ на стадии поставки, а затем команда становилась рыхлой и неэффективной. Это происходило потому, что члены команды начинали брать на себя неподготовленные задачи, искать задачи далее по бэклогу или еще хуже – топтаться на месте. Команде обычно достаточно двух-трех завершенных итераций, чтобы привести скорость подготовки бэклога в соответствие скорости поставки.

Стадия исследования завершается с выполнением двух больших задач, требующих подготовки, – *презентации результатов исследования* и планирования дальнейших действий (PI-планирование, если вы следуете SAFe). На презентации со всей организацией обсуждаются выводы стадии ознакомления. Она обычно представляет из себя одно- или двухчасовую сессию, на которой представляются результаты, полученные на стадии исследования. На планировании собираются команда, ответственная за поставку, и заинтересованные стороны для составления планов на несколько месяцев вперед. Акцент ставится на зависимостях между командами, установке базовых целей для каждой итерации, а также управлении релизами и совместными рисками (руководства по PI-планированию на сайте SAFe стоят вашего внимания). Вам нужно убедиться в том, что ваша организация готова и на уровне бизнеса, и на уровне ИТ. Вы хотите убедиться в том, что содержание для PI-планирования подготовлено и что у вас достаточно инструментов для проведения сессии планирования, особенно если требуется технология взаимодействия участников планирования, находящихся в различных местах [1]. Некоторым людям может быть знаком термин *big room planning* из практики бережливой разработки: он предполагает то же самое, только когда команды ведут планирование все вместе в большой комнате, что способствует сотрудничеству в рамках организации [2].

Когда вы завершите стадию исследования, команда, отвечающая за поставку решения, будет обладать насыщенным контекстом для работы. Это позволит команде и каждому ее члену в отдельности принимать решения, которые с высокой долей вероятности окажутся правильными. Возвращаясь к мотивирующим принципам Дэна Пинка, можно сказать, что мы предоставили командам автономию, включив их в процесс принятия решений на стадии исследования (особенно во время планирования и оценки, от которых они чаще отстранены при традиционных подходах), мы наделили их целью, когда наладили взаимодействие с бизнесом и заинтересованными сторонами и выделили время на понимание бизнес-проблемы и воздействия на клиентов. Контекст также важен для последнего аспекта: самосовершенствования.

Для самосовершенствования необходимы контекст и понимание области, в которой следует улучшать навыки. Одним из препятствий может стать так

называемый эффект Даннинга–Крюгера, который иногда называют эффектом ложного превосходства [3]. Эффект Даннинга–Крюгера возникает, когда вам неизвестно, как выглядит хороший результат (например, у вас нет правильного контекста для задания), но вы думаете, что довольно хорошо справляетесь. Этот эффект объясняет, почему мы порой слишком рано празднуем победу. Взгляните на организации, которые, как им кажется, стремительно развивают свою гибкость, но сколько стикеров с заметками висит в их офисах! При оценке зрелости я часто сталкиваюсь с тем, как команды с наименьшими показателями зрелости с воодушевлением рассказывают о выполнении задач, которых они в полной мере не понимают. При этом команды с наивысшими показателями зрелости видят множество возможных улучшений и оценивают себя как незрелые.

Я узнал об эффекте Даннинга–Крюгера методом проб и ошибок. Будучи консультантом, я часто просил провести оценку зрелости. При этом обычно полагаешься на результаты опросов, и при такой оценке можно проследить эффект Даннинга–Крюгера. Вот как это обычно происходит:

Я: Вы используете непрерывную интеграцию?
Разработчик: Да, используем.
Я [про себя: ладно, могу поставить тут галочку, но сначала проверю]: *Как именно вы используете непрерывную интеграцию?*
Разработчик: У нас есть Jenkins-сервер.
Я: И что вы делаете при помощи Jenkins?
Разработчик: Мы собираем наши приложения.
Я: Как часто Jenkins-сервер собирает ваши приложения?
Разработчик: Ну… он запускается раз в неделю, на выходных.
Я [ох!]: *Ладно, здесь нужно еще немного разузнать до того, как будет возможна какая-то оценка.*

Вместе с одним из моих клиентов мы поняли, что самостоятельное оценивание не предоставит нам нужных результатов из-за эффекта Даннинга–Крюгера, и разработали другой подход. Студентом я часами просиживал за игрой под названием Civilisation, и отсюда мы взяли идею и дерево технологий. Это дерево описывает различные технологии и последовательность, в которой они могут быть изучены (например, вам нужно изучить астрономию, прежде чем вам можно будет доверять ориентировку по звездам) [4].

В нашем дереве технологий непрерывной поставки мы предоставили контекст для команд, который описывает зависимости между практиками (например, непрерывная интеграция нуждается в автоматической сборке, а сборки запускаются после проверки и автоматизированного модульного тестирования). Это, в свою очередь, позволило командам охватывать взглядом все дерево технологий; следуя дереву, становилось намного легче совершенствоваться.

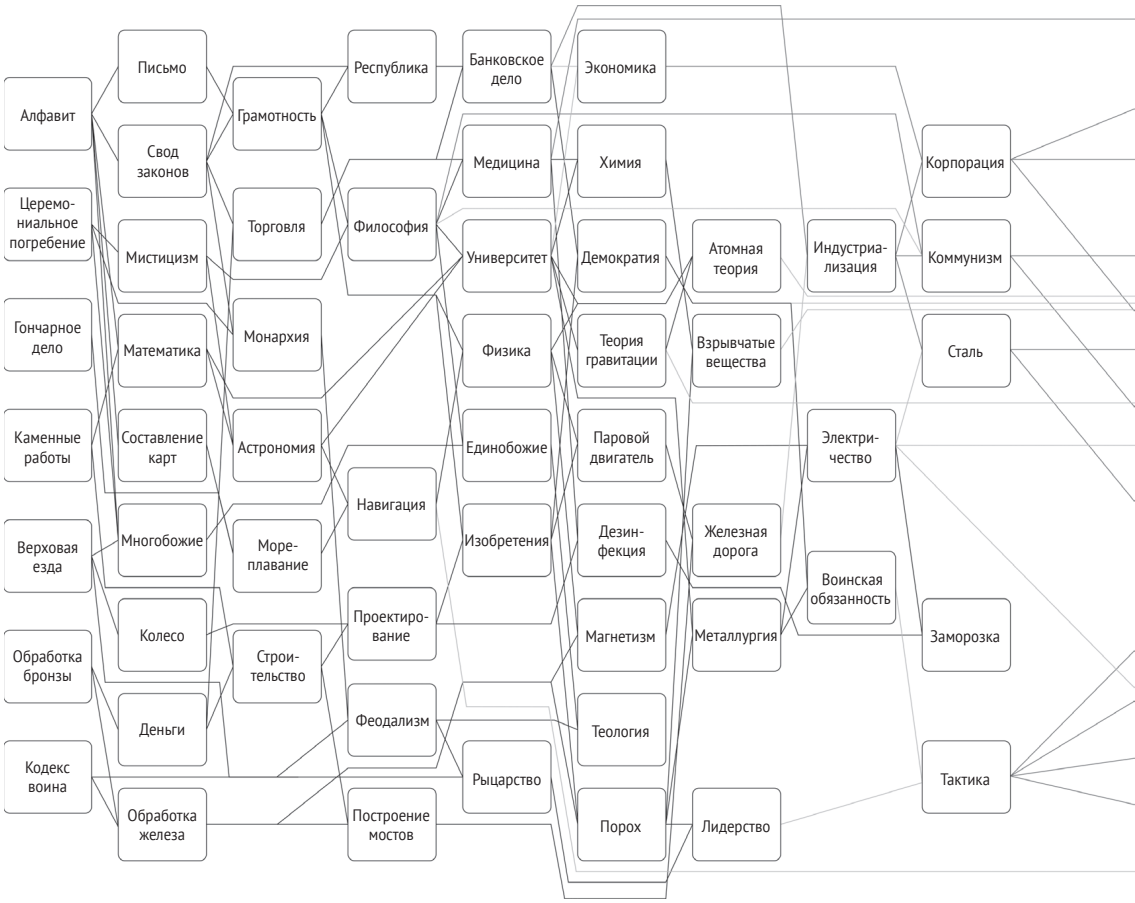


Рис. 5.2а. Дерево технологий:
пример, демонстрирующий зависимости между технологиями

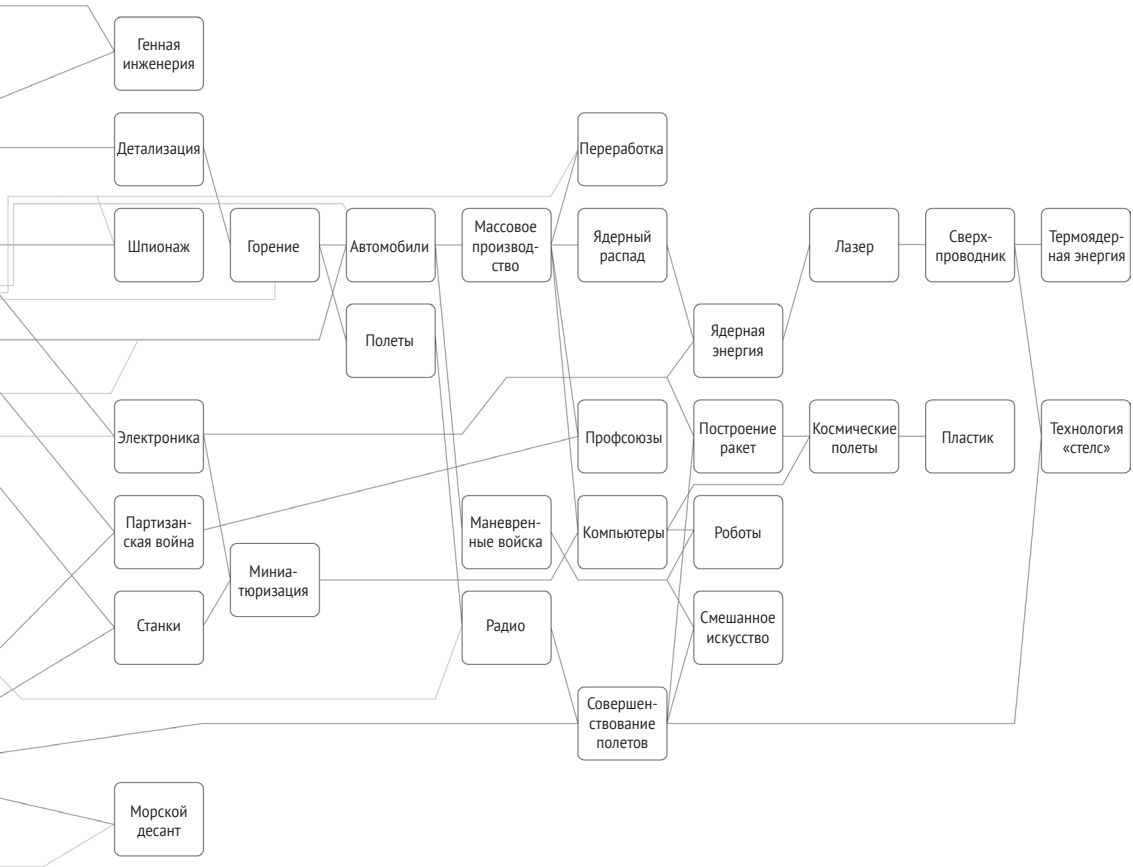


Рис. 5.2а (продолжение)

РОТАЦИЯ ФУНКЦИЙ
УПРАВЛЕНИЕ КОНФИГУРАЦИЯМИ
МОДУЛЬНОЕ ТЕСТИРОВАНИЕ
ПРАКТИКИ СБОРКИ
ПРАКТИКИ РАЗВЕРТЫВАНИЯ
Ротация функций на основе кода
Ротация функций на основе SCM
Внешнее управление ротацией
Централизованный фреймворк ротации функций
Централизованное управление ротацией функций
Переключение аудитории (для тестирования среды эксплуатации)
Все артефакты переключаются/ подвержены контролю версий
Разбиение кода на уровне IR
Разбиение кода на уровне функциональности
Слияние с применением инструментов
Автомати-зированное слияние
Динамическое слияние
Автоматизирован-ный фреймворк модульного тестирования
Покрытие модульного тестирования 5–25 %
Модульное тести-рование с моками и тестовым покры-тием в 25–80 %
Тестовое покрытие 80 %
Предварительные проверки качества
Комплексное регрессионное модульное тестирование
Сборка на основе комита или других триггеров SCM
Непрерывная интеграция
Стандартный процесс сборки
Автоматизирован-ная сборка (с регулярностью < 1 дня)
Быстрое тестиро-вание при сборке и приоритизация элементов сборки
Одна сборка – множество развертываний
Репозиторий сборочных артефактов Telstra
Выделенный сервер для сборки
Обратно совместимые интерфейсы
Ежедневные проверки
Автоматизирован-ные откаты
Автономное развертывание в предCAT-среде
Автономное развертывание в CAT
Автономное развертывание в среде эксплуатации
Непрерывная поставка
Автоматизирован-ные смоук-тесты после разверты-вания
Стандартизированное развертывание во все тестовые окружения до среды клиентского тестирования
Стандартизированное развертывание во всех средах
Автоматизированное развертывание в тестовые окружения до среды клиентского тестирования и проверка выполнения критериев качества
Автоматизированное развертывание в окружение для клиентского тестирования и проверка выполнения критериев качества
Автоматизирован-ное развертывание приложения
Автоматическое развертывание статических данных
Автоматизирован-ное развертывание базы данных

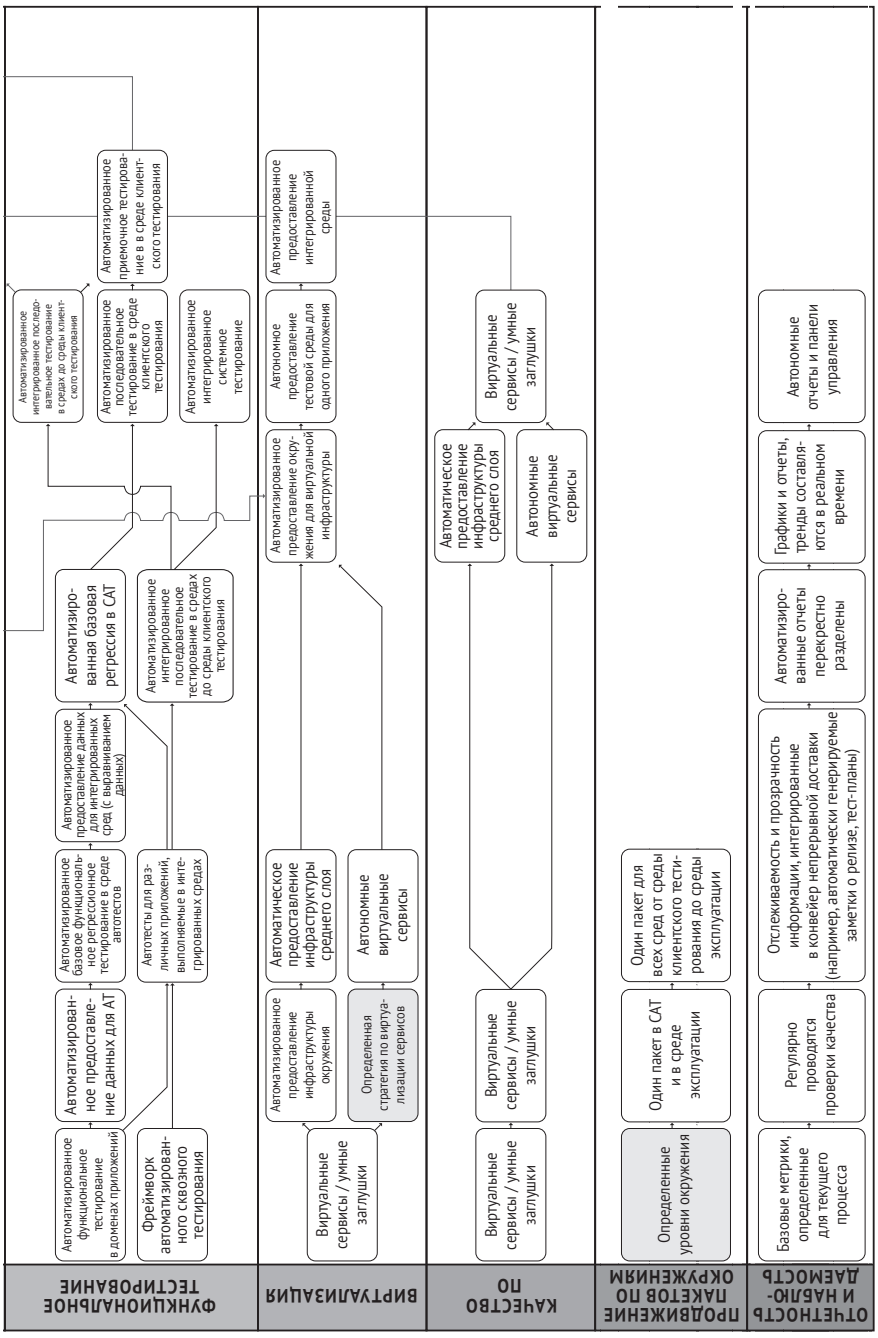


Рис. 5.26. Дерево зависимостей DevOps-технологий

Первые шаги вашей организации

Осуществляйте стадию исследования в рамках вашей инициативы

Я подробно описал стадию исследования в этой главе и рекомендую вам проводить такие сессии в вашей организации. Вот некоторая программа для двухнедельной стадии исследования, которую вы можете адаптировать для работы с крупными или малыми инициативами. Эти виды деятельности указаны для примера; существует множество других занятий, которые вы можете использовать в ваших сессиях исследования, чтобы усилить восприятие.

Часть I: изучение бизнес-проблемы (два дня)

- Брифинг от спонсора инициативы (с максимальным вовлечением в бизнес-проблему)
- Текущий опыт заказчика / заинтересованной стороны
- Представление проблемы глазами заказчика
- Формулировка миссии
- Определение критерия успеха
- Определение элементов, включенных и не включенных в проект
- Ознакомление с работой с заинтересованными сторонами
- Обсуждение схемы приоритизации
- Определение и смягчение рисков, проблем и зависимостей

Часть II: составление решений (три дня)

- Высокоуровневое предварительное проектирование
- Высокоуровневое составление технической архитектуры в поддержку предварительному проектированию
- Разделение объема работ на отдельные функции
- Определение минимально жизнеспособного продукта (MVP)
- Избирательные погружения в технологии и процессы

Часть III: планирование стадии поставки (пять дней)

- Обучение Agile (при необходимости)
- Определение структуры команды
- Оценка объема работы
- Высокоуровневое планирование
- Техническая подготовка (стратегии управления окружением, качеством и конфигурацией)
- Настройка управления доставкой
- Общественный договор
- Предварительное составление бэклога
- Представление стадии исследования организации
- Совещания по планированию (PI-планирование при помощи SAFe)

ГЛАВА 6

Структура, приносящая успех

> Никому еще не удавалось решить проблему
> с помощью реструктуризации.
>
> *Анонимный автор*

Несмотря на то что я очень люблю высказывание, вынесенное в эпиграф (хотя так много организаций, кажется, игнорируют этот посыл!), я думаю, что плохая организационная структура может оказать значительное влияние на вашу эффективность в рамках организации.

В наших устаревших организациях большая часть команды структурировалась по принципу функционального разграничения: команда бизнес-анализа, офис управления проектами, центр тестирования, команда разработки и команда администраторов. В вашей организации, возможно, к этому добавляется что-то еще. Тем не менее такой подход не идеален – отчасти согласно закону Конвея, который говорит о том, что в архитектуре вашего приложения будет отражаться структура вашей организации, или, если коротко, «если у вас четыре группы работают над компилятором, то вы получите четырехпроходный компилятор» [1].

На заре Agile мы узнали о силе колокации и о том, как помочь кросс-функциональным командам работать быстрее и более гибко. Agile подразумевает, что в идеальной команде есть все работники, чей вклад необходим для общего успеха, начиная со стадии бизнес-анализа до релиза в среду эксплуатации и последующей поддержки. Звучит, конечно, здорово, но, честно говоря, я никогда такого не видел на практике. Я работаю в крупных организациях, где это маловероятно – вам придется арендовать целый выставочный зал, чтобы уместить в одном помещении всех необходимых участников! Тем не менее существуют некоторые способы определить структуру организации, которая будет стремиться к идеальной кросс-функциональной команде, и ее может добиться любая организация независимо от масштаба.

Бережливое и Agile-мышление также показали нам ценность постоянных команд. В мире, управляемом проектами, мы запускаем проекты, которые являются конечными по определению и имеют конкретную цель. Как правило, в устаревших организациях по завершении проекта команда распускается, а ее участники переходят в другие проекты. В этих новых проектах новые участники команд изучают новый проект и предварительно готовятся, прежде чем смогут приносить пользу на новом фронте работ. Мы можем избежать задержек, задействуя постоянные команды, поддерживающие продукт или сервис. С помощью такого изменения можно сократить «выстраивание команд вокруг работы» до «передачи работы в руки существующих команд».

Такая перемена в подходе потребует внесения некоторых организационных изменений, самое тяжелое из которых связано с тем, что финансирование зачастую привязано не к командам, а к проектам. Так как же нам перейти от финансирования проектов к финансированию команд? Простого ответа здесь, к сожалению, найти не получится, решение будет даваться тяжело, и контекст специфический. Вот несколько напутствий, которые помогут вам на этом пути:

- финансируйте команды, основываясь на их потоке создания ценности. SAFe может предоставить структуру, в которой команды будут финансироваться исходя из их потока создания ценности;
- договоритесь с бизнесом о финансировании команды согласно циклу бюджета. Я работал с лидерами, которые вели с заинтересованными сторонами разговор в защиту долгосрочного финансирования команды;
- придержите некоторую долю средств для покрытия пробелов финансирования. Некоторые ИТ-лидеры, с которыми я работал, просили несколько завышенную оплату, чтобы подстраховаться на те времена, когда команда не будет полностью финансироваться проектами;
- применяйте творческий подход к наращиванию финансирования, исходя из работы, совершаемой командой. Самые вовлеченные владельцы продукта в итоге начинают ловко находить дополнительные источники финансирования в организации, выполняя работу для нескольких заинтересованных сторон и, следовательно, получая доступ к этим средствам.

Не думаю, что вы сможете быстро решить эту проблему, зато вы найдете начальные пути решения, которые помогут двигаться в этом направлении и займут место в вашем списке дел. Внесите показатель продолжительности совместной работы команд (или продолжительности жизни команд) в список изменений. Облегчите возможность командам оставаться вместе и быть продуктивными.

На рис. 6.1 показана организационная структура, которую я рекомендую клиентам для применения в качестве начала для структурирования своих команд. Я называю это «бургерной схемой организации», поскольку визуально она напоминает бургер. В дальнейшем я пройдусь снизу вверх по этим слоям «бургера» и объясню, как все они могут успешно работать вместе.

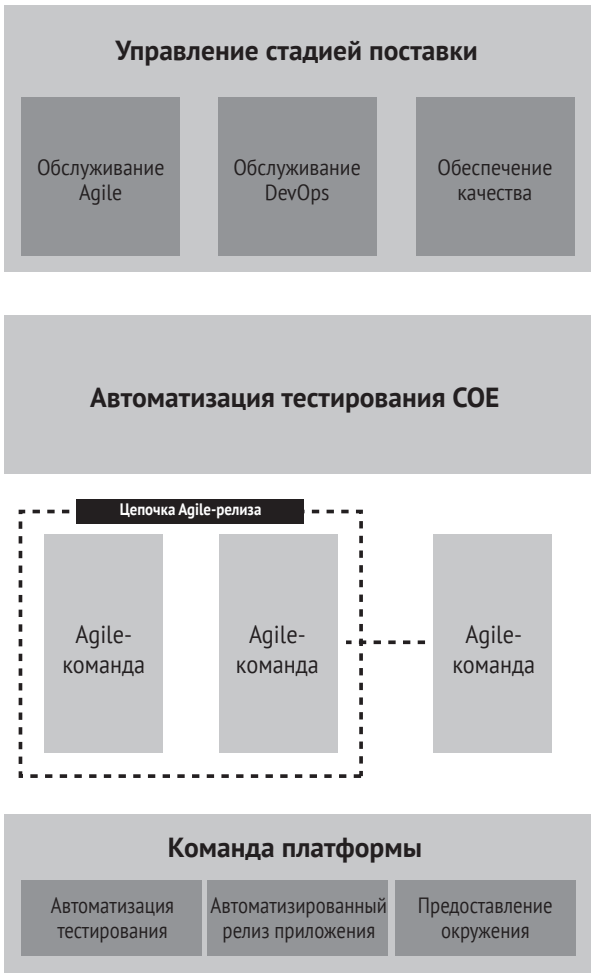


Рис. 6.1. Организационная структура начальной точки: эта многоуровневая структура прекрасно работает в крупных организациях

Команда платформы

Ведутся многочисленные споры о том, стоит ли иметь DevOps-команду. Некоторые дискуссии перерастают в холивары, поэтому я предпочитаю смотреть на результаты, прежде чем принимать решения, а не идеализировать взгляды на мир. Если вы посещаете саммит DevOps-корпораций либо другие Agile- или DevOps-конференции, то заметите, что организации, которые продвигаются в преобразовании своих усилий, чаще пользуются сервисом, выстроенным при помощи практик DevOps, или услугами DevOps-команд, или еще чем-либо подобным. В отчете о состоянии DevOps от 2016 года вы также можете отследить

множество людей, которые называют себя участниками DevOps-команд; соотношение менялось в течение 2015 и 2016 годов [2]. Руан Вильсенах называет создание DevOps-команд антипаттерном [3], поэтому пусть термин «DevOps-команда» вас не смущает, я просто называю это командой для платформы (можете использовать это название в вашей организации – я не обижусь). Команда для платформы ведет вашу разработку и поддерживает платформу для ваших приложений. В реальном мире эта платформа для разработки и запуска будет выступать в роли поставщика услуг, предоставляющего всем вашим командам возможности поддержки для поставки их решений[1].

В рамках команды для платформы вам нужно собрать людей, которые могут предложить весь ряд навыков, необходимых для DevOps. Это означает, что команда должна управлять инфраструктурой, конфигурациями, контролем качества, развертыванием приложения, мониторингом и, кроме того, релиз-менеджментом. Как вы видите, это кросс-функциональная команда экспертов, и мне кажется, что в нее войдут лучшие из ваших инженеров. Многие организации, которые работают с устаревшими системами, создавали структуры для обслуживания технологий (например, Windows-команда, UNIX-команда, Oracle-команда). Новой команде для платформы все же понадобятся навыки работы с этими технологиями, но внимание здесь больше будет уделяться навыкам автоматизации и способности переопределить процессы и архитектуры для текущих нужд. В итоге ваши команды для платформы не будут фокусироваться на какой-либо одной технологии.

Мы работали со страховой компанией, для которой смогли создать каталог услуг при помощи десятка стеков различных технологий (больше сорока услуг, в том числе развертывание и компиляция приложения). По каждой услуге мы позднее могли оценить прогресс, наблюдая за ее скоростью, надежностью и стоимостью. Это оказалось мощным инструментом для обсуждения некоторых технических улучшений с заинтересованными бизнес-сторонами.

Одна из проблем, с которыми я встретился в работе с командами для платформы, состоит в том, что они считают себя больше «стражами» DevOps, чем поставщиками услуг, что, в свою очередь, усложняет адаптацию команды к платформе. Зачастую масштаб перемен, которых требует менеджмент, недооценивается, что не способствует успешной адаптации. Когда платформа не способна предложить командам пути, которые позволят облегчить работу, команды начинают искать альтернативы этой платформе, создавая собственные решения. Сплоченность и гибкость в подходе нужны для того, чтобы команда для платформы могла работать с командами, которым она предоставляет услугу. Платформа должна правильные действия выполнять легче, чем неправильные. Также будет полезно подключить технических наставников

[1] Вы можете ознакомиться с обсуждением возможных организационных структур DevOps в статье блога Мэтью Скелтона «Какая структура команды поспособствует процветанию DevOps?». Структура, которую я здесь представил, подобна типу 4 – DevOps как модель обслуживания – в данном случае для внутренних команд [4].

к такой команде, чтобы они имели возможность работать с командами при возникновении проблем. Вы можете перемещать сотрудников в пределах команд разработки функциональности и команд для платформы или просто одолжить участников команды для платформы команде разработки функциональности: это поможет стирать культурные барьеры, которые в противном случае могут дать о себе знать.

Команда для платформы будет со временем развиваться. В исходном составе она, вероятнее всего, будет выполнять много задач, требующих ручного труда, а значит, это будет довольно большая группа, но она начнет уменьшаться по мере внедрения автоматизации. Чтобы иметь возможность внедрять автоматизацию все в больших масштабах, придется создать достаточный потенциал команды… Иначе будет как в анекдоте: «Мы слишком заняты для того, чтобы позволить себе совершенствоваться».

К обязанностям команды для платформы относятся следующие области в средах разработки, тестирования и эксплуатации:

- управление программными конфигурациями: все, что необходимо для обеспечения контроля версий приложения;
- процесс сборки приложения: скрипты сборки принадлежат команде приложения, но их выполнение и интеграция в цепочку автоматизированных инструментов входят в обязанности команды для платформы. Скрипты сборки часто создаются совместными усилиями, с тем чтобы они отражали требования обеих команд;
- интеграция с автоматизированным тестированием: подобным образом тестовые скрипты создаются командами приложения, но их выполнение запускается командой для платформы. Результаты представляет команда платформы во время запланированных запусков;
- предоставление окружения: команды вместе создают новые окружения и устанавливают необходимые программные пакеты;
- развертывание приложения: развертывание последней версии или пакета приложения тоже осуществляется совместными усилиями;
- мониторинг: текущий мониторинг инфраструктуры и приложений надлежит вести во всех средах;
- управление конфигурациями среды: требуется следить за тем, чтобы среды существовали в согласованной конфигурации, выявлять различия и вносить корректировки при необходимости;
- отчетность: включает в себя предоставление информации о состоянии платформы поставки и ее производительности.

Вы видите, что для того, чтобы полностью автоматизировать платформу поставки, нужно время, и управление текущим развитием команды платформы является ключевым аспектом вашего преобразования. Сомневаюсь, что проект получится осуществить очень быстро, так как еще не встречал никого, кто заранее знал все неизвестные факторы и точно оценивал изменения, связан-

ные с инструментами и технологиями. Структурированная карта и постоянное совершенствование помогут создать наилучшую платформу поставки для вашей организации.

Еще одна подсказка: команду для платформы необходимо обеспечивать возможностью управления изменениями. Технические специалисты склонны считать, что если однажды выбрать «правильные» инструменты, то все начнут следовать «правильному» процессу. Такой метод далеко не всегда работает. Вам нужен кто-то, кто будет способен создавать тренинги, вести обучение и общаться с пользователями платформы, чтобы получать «обратную связь» в ее отношении. У меня был один клиент, который после тренинга сказал, что ему больше не нужна помощь с DevOps-транформацией, так как у них уже установлены все необходимые инструменты, и все, что нужно, – это чтобы сотрудники ими пользовались. Конечно же, этот клиент позвонил мне несколькими месяцами позднее, чтобы обсудить, почему трансформация застопорилась! Примите мой совет: наймите сотрудников для управления изменениями для команды для платформы. Инженерам это понравится, так как они смогут получить помощь и необходимую документацию, которой им так или иначе пришлось бы заниматься самим.

Agile-команды

Самое интересное – это Agile-команды, которые проделывают большую часть работы. Я опущу то соображение, что некоторые команды поставки будут приверженцами методологии водопада и продолжат работать традиционным способом (существует множество литературы о том, как поставлять при помощи традиционных команд). Вместо этого сосредоточу внимание на Agile-командах поставки: для такого пути DevOps-трансформации потребуется провести определенные изменения в вашей организации.

Давайте начнем с обсуждения состава этих команд. Мы уже много раз упоминали кросс-функциональные команды и отмечали, что специалисты по эксплуатации должны вовлекаться в команду, чтобы достичь DevOps, в результате чего все больше специалистов будет появляться в постоянно растущей команде. На мой взгляд, непрерывное расширение команды – слишком простой подход, так как его не получится масштабировать в сложной среде, в которой требуется множество различных навыков в пределах Dev и Ops, необходимых для достижения результата. Вместо того чтобы раздувать команду, нужно сосредоточиться на выстраивании продукта и, в рамках этого процесса, на пользовании теми услугами других команд, которые являются неотъемлемой частью стадии поставки, такой как развертывание приложения. В Agile-командах должны присутствовать владелец продукта, бизнес-аналитик, скрам-мастер, разработчики, специалисты по качеству, тестировщики в соотношении, оптимальном для проектирования, сборки и тестирования системы. Со временем каждый участник команды должен будет нарастить навыки в различных направлени-

ях, чтобы выстроить Т-образный профиль навыков[1]. Мы доверим команде для платформы то, как приложение будет разворачиваться, и то, как должна выглядеть среда, чтобы приложение хорошо на ней работало. Agile-команды будут взаимодействовать с командой для платформы, но им не обязательно иметь в своем составе DevOps-инженера, когда понадобится большая поддержка; такого сотрудника можно позаимствовать из команды для платформы, на полную или частичную занятость.

Более того, в большинстве организаций в поддержке более поздних фаз тестирования будет участвовать внешняя команда по тестированию. Agile-команды должны будут привести систему к состоянию, наиболее близкому к релизу. Для того чтобы это происходило эффективно, Agile-команды будут пользоваться критериями готовности, чтобы определить, когда пользовательская история окажется готова к стадии поставки и будет достаточно детализирована, чтобы занести ее в бэклог спринта. Здесь используется критерий готовности (ready), позволяющий определить, когда можно завершить всю деятельность вместе с завершением спринта/итерации. Вот ряд типичных критериев готовности: история была измерена, рамки были обговорены, критерии приемки и тестовые случаи – определены. Критерии завершенности (done) включают как минимум следующее: код был разработан, функциональность задокументирована, история протестирована на модульном и системном уровнях, владелец продукта или представитель владельца просмотрели и приняли истории.

С позиции DevOps Agile-команды должны отвечать не только за поставку новой функциональности, но еще и за устранение любых дефектов в среде эксплуатации или за внесение мелких изменений в среду эксплуатации. Команды отвечают за приложение, которое они создают, а не просто передают его какой-то другой «команде по эксплуатации» или «команде по устранению проблем». Это мотивирует к созданию хорошего кода и значительно улучшает поддерживаемость приложения.

Данная перемена в пользу комплексных обязательств позволит Agile-командам более осмысленно подходить к бизнес-процессам. Лучше всего этого достигать за счет содержания Agile-команд, которые будут вести потоки ценности для бизнеса. Крупным организациям потребуется больше одной Agile-команды, и тут уже применяется концепция *цепочки релизов* SAFe для составления групп Agile-команд и управления ими с целью обслуживания потока ценностей[2]. В рамках этой группы Agile-команд необходимо уделить внимание ее технической композиции.

---

[1] Т-образные навыки предполагают знания и навыки во многих областях, притом что специалист более глубоко разбирается в одной из областей (в отличие от I-образных навыков, когда имеются знания лишь в одной из областей). Подробнее об этом говорится в приложении.

[2] Более подробная информация о цепочках релизов приводится на сайте ScaledAgileFramework.com.

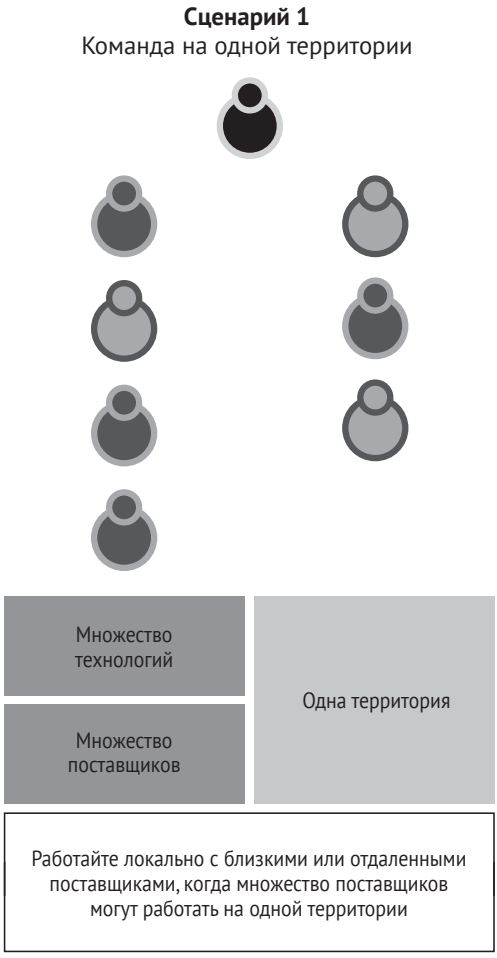


Рис. 6.2а. Сценарий 1 Agile-команды: Agile-команды по функциональности, находящиеся на одной территории, могут работать под началом различных поставщиков, а также со множеством технологий

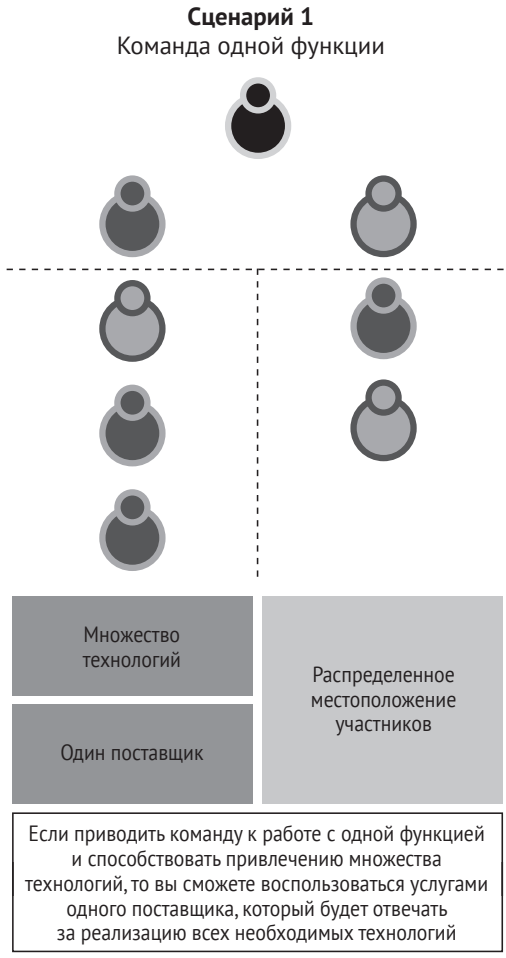


Рис. 6.2б. Сценарий 2 Agile-команды: распределенные Agile-команды по функциональности состоят только из участников одной организации-поставщика

Идеальным положением для вас будет то, что эти Agile-команды будут иметь возможность поставлять функциональность автономно. В Agile-командах по функциональности будут разработчики всех затронутых приложений, которые могут вести разработку, выходя за пределы одной системы. Данный принцип замечательно работает, когда разработчики работают в одной и той же организации, в одной и той же местности. Когда вы начнете проводить интеграцию из множества различных мест, я полагаю, что эта модель перестанет быть эффективной, так как для ее реализации потребуется усложнение организации

процессов, протекающих в пределах одной Agile-команды по функциональности, а распределенность команды усложнит создание культуры сотрудничества среди участников.

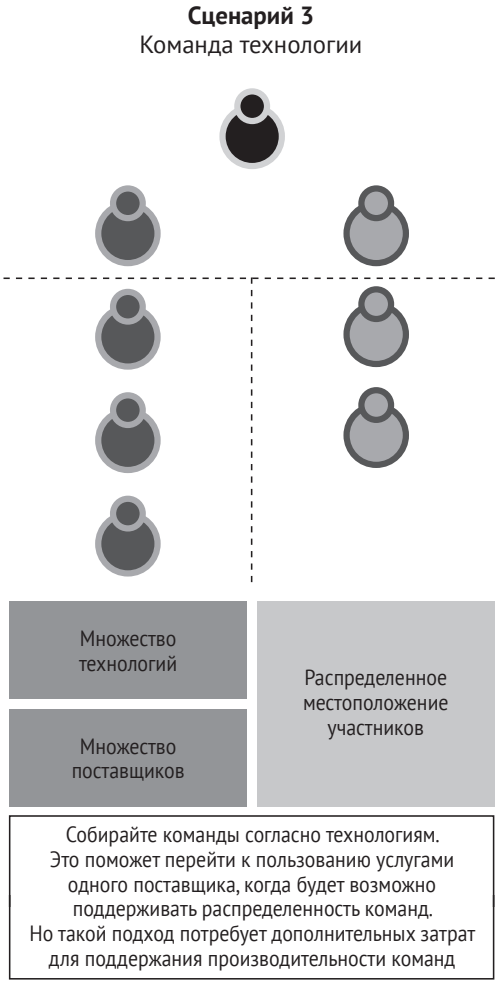


Рис. 6.2в. Сценарий 3 Agile-команды: Agile-команды по компонентам работают по сценарию наличия множества поставщиков и местоположений

При работе со множеством поставщиков вы вполне сможете воспользоваться подходом с Agile-командами по функциональности, если справитесь с тем, чтобы разместить всех участников команды в одном здании с поставщиками; близкое соседство (колокация) со временем сотрет культурные разногласия. Или можете попробовать обратную модель: просить ваших поставщиков ра-

ботать на вашей территории и под вашим руководством, что даст вам больше возможностей контролировать методы работы и культуру взаимодействия в среде команды продукта.

Если вы работаете только с одним поставщиком, который, в свою очередь, является вашим стратегическим партнером, то сможете с успехом создать распределенную многофункциональную Agile-команду, так как здесь будет присутствовать некоторая сплоченность благодаря организационным процессам, осуществляемым в пределах этих двух организаций. Вести работу со многими поставщиками, одновременно обслуживая множество технологий в распределенной Agile-команде по функциональности, кажется мне невероятно сложной задачей.

Альтернативной моделью выстраивания команд может послужить введение Agile-команд по работе с компонентами, которые будут осуществлять реализацию в пределах приложения или технологии. Для этого понадобится приложить больше усилий по управлению множеством команд, но этого можно достичь за счет общего планирования и синхронизации событий, например применяя PI-планирование, презентацию систем в SAFe. Это пока что не идеальная модель, исходя из моего опыта, но ею намного проще управлять в условиях работы со множеством поставщиков и из разных местоположений.

И последнее замечание насчет структуры команды, прежде чем мы перейдем к автоматизации тестирования. Рассматривать колокацию и распределенность, по моему мнению, необходимо с учетом того, чтобы найти оптимальный баланс между гибкостью/скоростью (в случае колокации команд) работы и ее стоимостью (в случае распределенных команд). Вам придется решать, что для вашей организации важнее. Обслуживание распределенных команд обойдется дороже, и поставка будет осуществляться медленнее, но существуют и высокопроизводительные распределенные команды, которые по обоим показателям объективно лучше, чем средняя команда, работающая в колокации. Здесь вам стоит руководствоваться имеющимся контекстом и вашими представлениями о командах.

Центр автоматизации тестирования

Автоматизация тестирования входит в состав жизненного цикла DevOps и, по моим наблюдениям, чаще других инициатив выдает неудовлетворительные результаты. Тем не менее автоматизированное тестирование выступает предпосылкой многих достижений, которые в перспективе сулят отменные результаты. В рамках высокоуровневой организационной структуры я представляю центр автоматизации тестирования в качестве переходной стадии. Думаю, большинству организаций потребуется такой центр, для того чтобы ряд специалистов сосредоточился на работе над автоматизацией тестирования и на создании цельного фреймворка автоматизации тестирования. Это входит в список обязанностей центра автоматизации тестирова-

ния, в котором работают тест-архитекторы и инженеры по автоматизации тестирования, ведущие разработку этого фреймворка автоматизации тестирования. Со временем данная обязанность и команда будут распределяться, и автоматизация тестирования станет поддерживаться тест-инженерами и архитекторами из команды самого приложения. Они будут поддерживаться живым практикующим сообществом, в котором фреймворк автоматизации тестирования обслуживается в стиле open-source. И конечно же, организация может решить, что эта команда должна присутствовать на постоянной основе. Учитывая, что такие команды зачастую намного меньше в сравнении с персоналом центра тестирования, их содержание может оказаться относительно недорогим.

Команды по обслуживанию и обучению

Я уже ранее говорил об определении процессов бережливого управления в организации, которые основываются на объективных показателях, а не на субъективных концепциях. И естественно, такое управление должно иметь свою «обитель», воплощенную в виде команд по управлению и обучению. В самом начале трансформации, когда изменения в вашей команде будут происходить довольно часто, такие команды будут большими: ведь в их составе будут ваши организационные тренеры, тренеры по Agile и те, кто обслуживает процессы трансформации. Количество этих сотрудников будет сокращаться, по мере того как уменьшается частота изменений. Давайте поговорим о различных аспектах управления, с которыми такая команда должна иметь дело.

Обслуживание архитектуры

Совершенно очевидно, что архитектура вашего приложения будет непосредственно влиять на то, насколько быстро ваша команда сможет осуществлять поставку. Поэтому в рамках обслуживания архитектуры мы будем преследовать три цели:

1) убедитесь, что каждая инициатива все больше изолирует элементы архитектуры приложения, отчего оно становится более гибким, когда создаются более гибкие интерфейсы между приложениями и когда разработка приложения следует модулярному принципу построения архитектуры;
2) убедитесь, что каждая новая инициатива устраняет все больше технических недоработок в системах;
3) наблюдайте за эволюцией архитектуры приложения, следите за тем, чтобы она развивалась в соответствии с видением компании, и за тем, чтобы приложение продолжало соответствовать потребностям бизнеса. (Мы вернемся к этому, когда будем обсуждать сценарии, по которым может развиваться архитектура.)

Важно заметить, что обслуживание архитектуры нужно осуществлять в тесном взаимодействии с командами приложения, чтобы избегать разработки подходов, которые затруднительно реализовать.

Обслуживание методов

Существует множество различных методов поставки, которыми сегодня могут пользоваться организации. Подходы к обслуживанию методов помогут убедиться в том, что для каждой инициативы используется оптимальный метод. Также это подразумевает, что команды должны учиться применять соответствующие методы и поддерживать актуальность методов внутри компании, чтобы этот фреймворк люди могли использовать массово. На личном опыте я понял, что разрастание терминологии и практик при отсутствии общего фреймворка приводит к появлению неровностей в работе, когда вы начинаете перенимать и разворачивать методы Agile.

Обслуживание качества поставки

Ускорение выхода на рынок – наша основная цель, но мы все так же должны преследовать ее, не подвергая рискам качество продукта. Обслуживание качества поставки ориентируется на показатели жизненного цикла разработки, чтобы выявить проблемы с качеством и улучшить общее качество продукта. При этом в длительной перспективе не должно оказываться воздействие на скорость или стоимость поставки.

Обслуживание непрерывного совершенствования

Как я упоминал ранее, эффективное управление непрерывным совершенствованием является ключевым элементом трансформации в целом. В отношении каждой из крупных инициатив это обеспечит необходимые критерии успеха, основания этих критериев и последующую оценку успеха. Владелец или менеджер, ответственный за финансирование непрерывного совершенствования, сможет поддерживать наиболее многообещающие инициативы, следуя подходу WSJF (Weighted Shortest Job First). Согласно WSJF-приоритизации, инициативы, схожие по уровню окупаемости, приоритизируются по их масштабности. Чем меньше инициативы, тем выше их приоритет, что ускоряет циклы обратной связи [5].

Но как же проектные менеджеры?

Вы могли бы спросить, почему в моей организационной структуре нет отдела проектного менеджмента и почему я не упоминаю проектных менеджеров, говоря о команде. Я не сторонник мнения о том, что проектный менеджмент – пережиток прошлого. Полагаю, что для присутствия менеджеров

на проекте имеется масса причин. Тем не менее для описываемой структуры постоянных команд проектные менеджеры не совсем подходят. По определению, проект существует ограниченный период времени, в течение которого он должен достичь определенной цели. Таким образом, проектные менеджеры будут отвечать за обслуживание поставки продуктов, пользуясь данной структурой команды. В конце концов проектов должно становиться меньше, так как работа в командах продвигается вместе с рабочими элементами Agile (эпиками, функциональностями или историями), а в таком случае будет меньше проектных менеджеров. Но для ведения больших и сложных проектов вам может понадобиться нанять проектных менеджеров, чтобы они обслуживали поставку в рамках всех команд и составляли отчетность по всей работе, связанной с проектом, – это то, что сложно было бы осуществить силами команд поставки, которые больше сосредоточены на работе с бэклогом. Соответственно, проектный менеджер будет активно вовлечен в процессы, выходящие за рамки проектного планирования и подразумевающие внесение вклада в Agile-планирование от лица заинтересованных лиц.

Первые шаги вашей организации

Определите один из потоков ценностей и команду, работа которой будет связана с ним

Мы уже говорили о создании схемы потока ценностей ранее (в главе 2). Здесь мы рассматриваем описание потока ценностей с точки зрения бизнеса. Как только вы опишете поток ценностей, следующим шагом будет определение систем, которые будут обслуживать поток ценностей.

Вглядываясь в свой бэклог работы или в портфолио инициатив, определите, насколько работа повлияет на системы, обслуживающие поток ценностей. Основываясь на этой информации, вы сможете создать структуру команды, которая сможет заниматься потоком ценностей. Она не будет идеальной, и со временем вы ее отладите, но это дает вам отправную точку. Согласно терминологии SAFe, на данный момент вы определили цепочку релиза Agile и теперь можете продолжить поставлять работу при помощи такой структуры команды, обслуживающей поток ценностей. Со временем вы смените модель финансирования, чтобы больше поддерживать команды, как было показано выше в этой главе.

Определите команды, которые будут затронуты при переходе к команде платформы

Команда платформы – это концепция, подразумевающая множество преобразований, и необходимые изменения зачастую недооценивают. Чтобы помочь вам ориентироваться в этих изменениях, предлагаю вам обозначить команды, выполняющие на данный момент функции, которые будут выполнять команды платформы или на которые повлияет работа команд

платформы (например, команды инфраструктуры, команды по тестированию, администраторы баз данных [DBA]). Пригласите этих сотрудников на общий тренинг, чтобы поговорить о том, как должна выглядеть платформа поставки в вашей организации с функциональной точки зрения. Когда у вас будет достигнута договоренность, обсудите, как должна обслуживаться платформа поставки. К счастью, команда платформы развивается по мере появления договоренностей внутри команды. После этого договоритесь о последующих шагах, чтобы еще больше приблизиться к конечному общему видению.

ГЛАВА 7

Из тестировщиков в инженеры по качеству

> Хорошие тестировщики смотрят в обе стороны, перед тем как переходить дорогу с односторонним движением.
>
> *Из жизненного опыта тестировщиков*

В предыдущей главе я говорил об организационной структуре вашей ИТ-трансформации, и в моей команде присутствовали аспекты тестирования. Почему я посвятил еще одну главу тестированию? По моему опыту, организационные перемены, связанные с тестированием, зачастую невероятно важны и одновременно трудоемки для организации. Изменения же для тестировщиков куда более масштабны – работа, которая была сосредоточена на выполнении тестовых скриптов, вылилась либо в очень технологичное проектирование, либо в оценку рисков и выстраивание стратегии проектирования, учитывающей эти риски. Каждый день я встречаюсь с клиентами, которые все еще предпочитают иметь отдельные отделы тестирования, сосредоточенные на традиционном тестировании вручную (которое выполняется в недостаточной мере и отвечает лишь требованиям по оптимизации стоимости, а не скорости работы или рисков), и мне бы хотелось, чтобы мы могли это исправить. Довольно часто я наблюдаю неудачные или непродуманные решения автоматизации тестирования у клиентов, которые являются приверженцами отделов тестирования и менеджеров тестирования. Переход от обычного тестирования к QE качественной инженерии потребует изменений в мышлении людей, занятых в тестировании. Не все смогут справиться с этим переходом. И хотя эта глава описывает организационные моменты, я раскрою и некоторые технические темы, важные в данном отношении.

Организация обеспечения качества

Качество – это важная область, которая находит отражение во всех частях организации. На рис. 6.1, на уровне обслуживания, определяются стандарты и принципы, которые служат обеспечению качества. Вы также будете ежедневно оценивать процессы обеспечения качества продукта, и наблюдение должно стать частью процесса управления. Это значит, что показатель качества и итоги ретроспективы и ревью нужно обсуждать для того, чтобы определить, как улучшить процессы обеспечения качества и где стоит отредактировать стандарты.

Изначально команды по автоматизации тестирования требовались для установки фреймворка автоматизированного тестирования; кроме того, они работали с командой платформы при выстраивании правильной интеграции, которая позволяла бы выполнять целенаправленное автоматизированное тестирование (например, тестирование только изменяемой функциональности). Главная задача команды состояла в том, чтобы, подключив всех инженеров по тестированию, успешно масштабировать фреймворк автоматизированного тестирования; следовательно, для ознакомления с фреймворком командам требовалось достаточно много времени уделять наставничеству и парному программированию. Команде также необходимо создать некоторый каркас, чтобы автоматизация тестирования со временем не пришла в упадок. Этот каркас поможет убедиться в том, что выполнение тестирования не будет занимать слишком много времени и не потребует слишком много ресурсов (яркий пример – усугубление обеих проблем при длительном ведении автоматизированного тестирования). Этим каркасом могут выступать стандарты написания кода, общие библиотеки, регулярное peer-review рецензирование или другая техническая документация либо процессы обслуживания. На ранней стадии все будет протекать довольно интенсивно, особенно по мере того, как фреймворк автоматизированного тестирования будет развивать поддержку всех потоков данных. Позднее команда сможет сократить размеры, когда инженеры по тестированию уже ознакомятся с фреймворком и им больше не понадобится серьезная поддержка. И хотя мои клиенты стараются оставить команды по автоматизированному тестированию в качестве постоянной составляющей, я вполне могу себе представить, что такая функция перестанет существовать в виде отдельной команды и будет переложена на ряд наиболее опытных инженеров по тестированию.

Выше мы уже говорили о командах платформы. Просто напоминаю, что эта команда будет интегрировать скрипты, чтобы весь конвейер обеспечивал адекватное поведение и выводил результаты. Команда платформы также работает с инженерами по тестированию для согласования стандартов, которые обеспечат должную работу скриптов в платформе (например, переменные окружения).

А затем, конечно же, инженеры по тестированию в командах поставки будут иметь скрипты автоматизированного тестирования и обеспечивать покрытие всех функциональных областей. Они работают со всеми остальными командами – так они всецело отвечают за качество сервиса.

Процесс обеспечения качества

Давайте обсудим распространенное ложное представление об автоматизации: мол, это просто автоматизация тестов, которые вы так или иначе выполняли бы вручную. Это не так! И возможно, не в последнюю очередь по этой причине инициативы по автоматизации оказываются неудачными: в них пытаются автоматизировать лишь ручные тесты. В редких случаях это может сработать, но в большинстве случаев такой подход заводит в тупик. Мне больше нравится термин QE («качественная инженерия»), чем «автоматизация тестирования», потому что в нем не упоминается слово «тест», которое может ассоциироваться с отчаянной попыткой найти все баги в конце жизненного цикла разработки. Проектирование качества связано с комплексным аспектом автоматизации, который обеспечивает качество конечного продукта. Это означает, что человек или команда, отвечающие за обеспечение качества, должны взглянуть на весь жизненный цикл, не только на фазу тестирования – а это довольно сложная задача. Автоматизация тестирования (например, автоматизированное выполнение тестовых скриптов) – деятельность, связанная с обеспечением качества.

Традиционный центр тестирования развивался как противоположность команде поставки; он контролировал качество продукта, прибегая к массовым проверкам. Это вызвало нарушение нормального функционирования, были упущены возможности для улучшения качества, и зачастую команды спорили о том, кто должен нести ответственность за упущенные проблемы качества (тестировщики, не обнаружившие проблем, или же создавшие их разработчики). Можно стать на сторону разработчика и сказать: «Ничего страшного, что здесь проблема: на то и есть тестировщики, чтобы ее решать, это их работа». С другой же стороны, тестировщик мог бы сказать: «Я не должен принимать это близко к сердцу, так как хороший разработчик не создает проблемы намеренно». Вам нужно изменить такое отношение и сделать качество общей целью. Деминг говорил, что «качество произрастает не из всеобъемлющих проверок, а из постоянного улучшения качества процесса производства» [1]. Это значит, что тестер, который небезразличен к проблемам, сможет вносить больший вклад в процесс поставки – не как соперник, а как член команды.

Также проблемы существуют и в измерении качества: многие организации пользуются дефектами или частотой возникновения дефектов как средством измерения качества[1]. Нет оснований считать, что качество улучшилось, основы-

[1] Я довольно скептически отношусь к измерению качества продукта по количеству дефектов. Мне привелось общаться с представителями компании в США, которая проходила программу ИТ-совершенствования по причине того, что у них было много дефектов в цикле разработки. После того как они сообщили ИТ-отделу, что количество дефектов будет выступать показателем качества, оно вдруг резко сократилось. Когда дефектов стало совсем немного, организация посчитала, что наступило время для снижения темпов тестирования. Но еще до того, как это решение было принято, количество дефектов снова начало расти. Лид по процессу преобразования позвал команды поставки и тестирования к себе в кабинет, чтобы ему объяснили, что происходит. Ответ нашелся в ретроспективе: когда дефекты являлись основным крите-

ваясь только на количестве дефектов. Я надеюсь, вы согласитесь со мной в том, что нам нужны более подходящие способы улучшения качества, поэтому давайте обсудим, что можно осуществить и как мы можем это измерить.

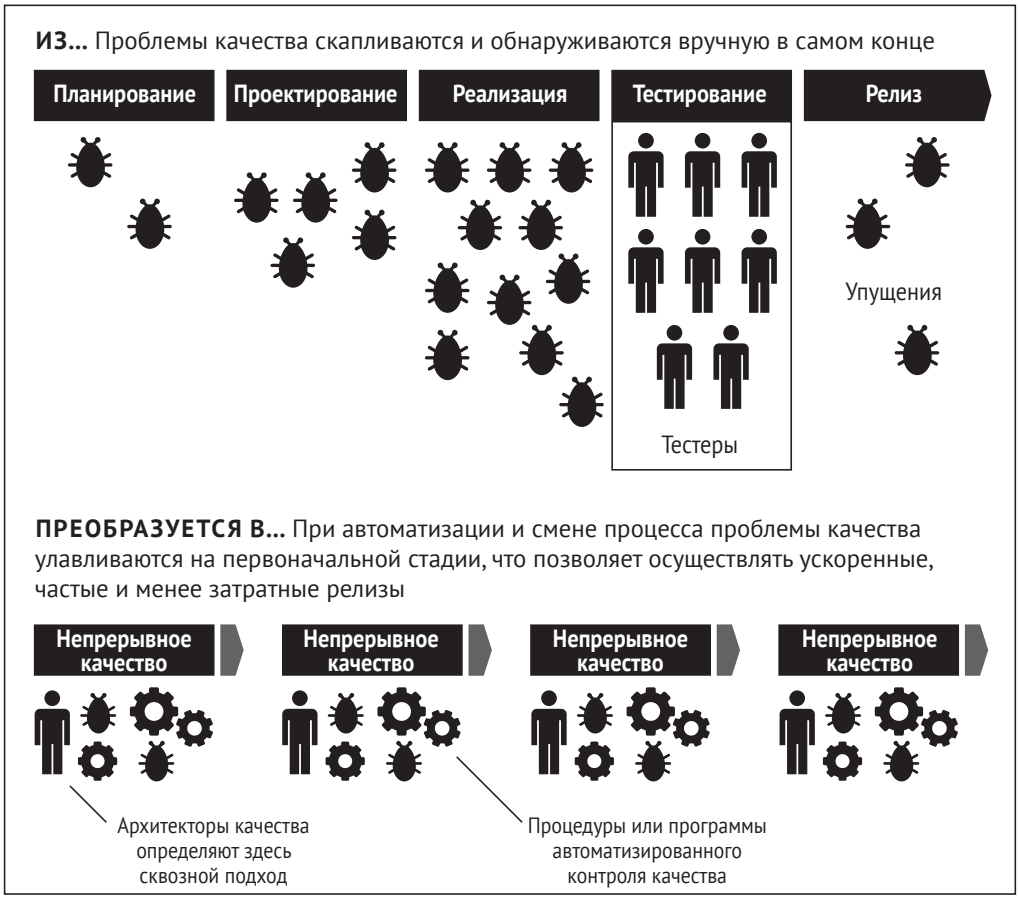


Рис. 7.1. Процесс обеспечения качества:
при переходе от тестирования к обеспечению качества
процесс сосредоточивается на всем жизненном цикле продукта

рием успехов в разработке, команда разработки тесно сотрудничала с командой по тестированию и вся функциональность проходила «предварительные тесты» перед формальным тестом, а любые проблемы быстро исправлялись. Даже был заведен новый тип тикета в трекинговой системе: «предварительный запрос о дефекте». Когда сотрудники отдела тестирования услышали о том, что их будут сокращать при понижении уровня качества, они начали отчитываться о большем количестве дефектов и меньше о «предварительных запросах». Лид ИТ-трансформации сказал мне, что на данный момент качество процесса поставки вовсе не поменялось в процессе прохождения, и все изменения касались лишь процесса поддержки.

За процессом обеспечения качества (QE) стоит идея о том, что вам нужно обеспечить качество во всем, что вы делаете, и искать способы проведения дополнительного тестирования, не подразумевающего дополнительных затрат (как средств, так и времени). Фреймворки наподобие SAFe и строгой Agile-поставки (англ. *disciplined Agile delivery*, сокр. DAD) переняли эти принципы. Но что это означает на практике? Давайте разделим процесс на пять фаз:

1) требования, проектирование и архитектура;
2) написание кода;
3) Agile-тестирование;
4) стабилизация;
5) выпуск в среду эксплуатации.

Требования, проектирование и архитектура

Качество начинается с истоков, поэтому нам нужно иметь подход к качеству уже на ранних фазах проектов (Agile в том числе), чтобы осуществлять последующую оценку масштаба работ. На данном этапе мы можем предпринять несколько действий. Многим проектам недостает качественно написанных требований / пользовательских историй, и это архитектура на ближайший период. Однако с этим можно справиться. Конечно, здесь можно обнаружить области, касающиеся бизнеса и требующие вовлечения в работу экспертов, но остальное можно охватить при наличии хороших инструментов.

В компании Accenture используется инструмент, который анализирует требования и пользовательские истории и выявляет неточности в формулировке. Неоднозначные предложения выделяются цветом, например: «Время отклика должно быть *коротким*», «Должна быть обеспечена *легкая* навигация по экрану» или «Необходима поддержка *нескольких* различных способов оплаты». Вы можете создать свой инструмент или поискать бесплатное решение и научить вашу платформу распознавать неоднозначную речь. С внедрением все больших объемов искусственного интеллекта в жизненный цикл ИТ вы все больше будете замечать продукты, способные не только распознавать ключевые слова.

С точки зрения архитектуры, здесь присутствует качественная сторона, которая при проектировании должна учитываться. Вы должны быть уверены в том, что каждая инициатива (изменение) делает вашу архитектуру лучше. *Разделение систем* и модулей должно быть одним из ключевых элементов, к которым стремятся эти изменения. Как обсуждалось ранее, не существует законченной архитектуры, которую архитекторы должны придумать и попытаться воплотить; сегодня их задача в том, чтобы архитектура была подвижной и гибкой, для того чтобы в будущем ее можно было бы постоянно развивать, прикладывая все меньше усилий.

При традиционных подходах накапливался технический долг в архитектуре, и вносить изменения становилось все тяжелее. Следовательно, производительность ваших архитекторов необходимо оценивать по тому, насколько

гибкой становится архитектура. Например, как архитекторы поймут, что ваша архитектура выдержит испытания будущего? Я приведу несколько показателей архитектуры, которые можно измерить, и настоятельно рекомендую это сделать, чтобы у вас была какая-то опора, которая поможет вам стремиться к правильным процессам. Например, стоит обратить внимание на количество запросов на доступ к базе данных, объем данных, передаваемых по интерфейсам, *вызовы без состояния* и *вызовы с сохранением состояния*, количество вызовов ко внешним системам, ответы и время центрального процессора. Как правило, с течением времени это начнет поставлять вам интересную и полезную информацию о степени зрелости вашей архитектуры.

Важной составляющей обслуживания архитектуры выступает наличие ряда четко определенных архитектурных принципов. Каждый в организации должен знать, какое значение архитектура имеет в вашем контексте. Вам не обязательно этим увлекаться, как в Amazon (где, как ходит анекдот, Джефф Безос обязал всех ориентироваться в работе на потребности сервиса под угрозой увольнения [2]), но все же наличие возможности выбора из нескольких вариантов при принятии решений поможет соединить все подвижные компоненты. Как вы видите, мы пытаемся создать культуру сотрудничества и отслеживаемую среду для всего жизненного цикла стадии поставки; мы проводим разделение функциональных частей, не перестраивая при этом меры контроля, а накладывая их друг на друга.

Написание кода

Качественные аспекты написания кода обычно недооценивают. Я разговаривал со многими организациями, и немногие из них знали или беспокоились о том, какие еще меры обеспечения качества можно применять к фазе написания кода, и полагались только на фазу тестирования. При работе с интеграцией систем часто возникают разногласия по поводу мер обеспечения качества при написании кода. Среди них есть те, которые обязательно должны присутствовать: статический анализ кода, автоматизированное модульное тестирование и проверка кода коллегами. У вас должен быть фреймворк автоматизированного модульного тестирования, который будет использоваться для тестирования модулей непосредственно разработчиком в автоматизированной манере. Такие фреймворки были разработаны для многих технологий, например jUnit для Java или nUnit для .NET, – и нет оправдания таким случаям, когда они не используются и разработчики не могут снизить некоторую нагрузку от ручного тестирования. В дополнение к этому статический анализ кода реализуется довольно просто, даже для технологий в роде COTS-пакетов. Ваши стандарты написания кода, по большому счету, можно систематизировать и распространять таким образом.

Ручной просмотр кода коллегой, применяемый во многих организациях (хотя зачастую невероятно неэффективно), должен осуществляться только после автоматизированного статического анализа кода и модульного тестирования. Не

нужно проводить ручные проверки, тем более что это дорого и долго, – делу поможет автоматизация. Проверки кода коллегами должны сосредоточиться на архитектуре и концепции кода, а также должно проверяться соответствие его назначению (например, решает ли этот фрагмент кода проблему?).

Я был в некоторых организациях, которые с течением времени добавляли в список проверок кода коллегами новые пункты при обнаружении каких-либо проблем. А это привело к тому, что в какой-то момент список проверок разросся до 150 пунктов. И знаете что? Никто в самом деле не следовал этому списку, потому как он был слишком длинным и с каждым днем в нем находилось слишком много несоответствий. Создайте простой и компактный список проверок, который будет выступать в качестве руководства, и объясните сотрудникам, что искать, разбивая их на пары с одним более опытным разработчиком на время проверки.

Также вы можете группировать разработчиков для осуществления дружеских проверок ряда изменений на постоянной основе, чтобы они могли учиться друг у друга и придерживаться стиля написания кода, принятого в команде. Постарайтесь не делать процесс проверки коллегой более утомительным, поручая коллеге найти изменения самостоятельно и предоставить отзыв в таблице Excel. Это никому не облегчит работу. Лучше попросите оставить отзыв непосредственно в контексте пользовательской истории в той же системе управления работой, которой пользуется вся команда.

Agile-тестирование

Независимо от того, используете вы формально метод Agile или нет, функциональное тестирование осуществляется наилучшим образом, когда тестировщики и разработчики сидят вместе. Проектировщики тестов должны писать код для автоматизации тестирования одновременно с написанием функционального кода разработчиками. Когда те и другие работают вместе, им намного легче писать тестируемый код, проверку которого можно автоматизировать. Это один из важнейших приемов, позволяющих сделать автоматизацию тестирования более успешной и доступной по стоимости.

Если у вас есть команда автоматизации, которая владеет фреймворком, то тестировщики выгодно им воспользуются во время разработки, предоставляя обратную связь команде фреймворка. К тому же тестировщики будут работать с командой платформы с целью интеграции нужных тестовых скриптов, чтобы во время автоматизированных процессов сборки и развертывания выполнялись только необходимые скрипты. Больше всего нам здесь нужна быстрая обратная связь, а не полное покрытие. Мы можем выполнять полный ход проверок в нерабочее время: на выходных или ночью.

Во время разработки надлежит, помимо прочего, проверять производительность и безопасность. И хотя они могут быть не такими же, как в среде эксплуатации или в предшествующих средах из-за ограничений, мы можем принимать быстроту реакции за показатель. На самом деле мы предпочитаем скорость, а не

точность. Если мы находим дефекты 70 % времени на ранних стадиях жизненного цикла и на более поздних стадиях 30 % времени потратим на оставшиеся дефекты, это нормально. Помните: мы автоматизируем все аспекты проверок качества, поэтому сможем выполнять их намного чаще. Что касается производительности, мы можем не получить здесь точных результатов, но если выполнение действия со временем замедляется, мы знаем, что у нас есть нечто, на что стоит взглянуть и что может привести к дальнейшим проблемам.

Быстрая обратная связь позволяет разработчикам пробовать различные подходы улучшения производительности, все еще оперируя контекстом, вместо того чтобы заниматься этим неделями позднее, когда тестирование производительности проходит неуспешную проверку, а разработчик уже выстроил целое решение, основанное на первоначальной идее проекта. То же самое применимо к тестированию безопасности: добывайте обратную связь по возможности раньше.

Стабилизация

Как бы я ни хотел, чтобы весь процесс тестирования производился Agile-командой, это зачастую невозможно осуществить по ряду причин. Поэтому мы используем стадию стабилизации для выполнения тестов, которые связаны со внешними системами, занимают много времени или выполняются третьей стороной для проверки безопасности. Эта фаза стабилизации происходит прямо перед передачей в среду эксплуатации. Лично я считаю, что не мешало бы провести тестирование бизнес-сценариев и в рамках тестирования критериев приемки, а это не получится автоматизировать. Вам не нужно, чтобы ваши заинтересованные лица проводили тестирование по уже пройденным тест-кейсам; вместо этого нужна оценка удобства пользования и соответствия назначению.

Выпуск в среду эксплуатации

Сейчас я скажу самое основное, так как о поддержании работоспособности системы речь пойдет ниже, а пока достаточно отметить, что некоторые процессы обеспечения качества должны иметь место и в среде эксплуатации. Вам нужно не только отслеживать ваши серверы и сервисы, но и добиваться надлежащей производительности, доступности и функциональной точности в среде эксплуатации. Все это обязано стать частью полноценного плана по обеспечению качества.

Пара слов об автоматизации функционального тестирования

Как я говорил ранее, автоматизация тестирования является одним из видов деятельности, связанных с DevOps, который зачастую оборачивается для орга-

низации существенным испытанием, требуя от непосредственных участников принципиально другого образа мысли. Но все же я встречал некоторые общие признаки неудачной автоматизации тестирования и перечислю их, чтобы вы избежали соответствующих недочетов в своей работе.

Не надо недооценивать влияние инфраструктуры и экосистемы

Существует физическое ограничение тому, какое давление некоторое количество мануальных тестировщиков способно оказать на ваши системы; автоматизация применяет совершенно другой подход для тестирования вашей системы. То, что вы ранее делали вручную раз в неделю, теперь благодаря автоматизации можно делать сотню раз в день. Если прибавить к этому интегрированную среду, то вашим внешним системам понадобится еще и быстрее реагировать. Поэтому стоит обдумать два важных вопроса: может ли инфраструктура в ваших средах поддерживать объем тестирования, в сотню раз больший, чем сейчас, и настроены ли внешние системы на поддержку такого объема? Вы, конечно, всегда можете сократить нагрузку на внешние системы, ограничивая текущие операции взаимодействия и блокируя некоторый процент операций, или же воспользоваться виртуальными сервисами.

Не надо недооценивать недостаток данных

Очень часто автоматизированные тестовые скрипты используются в той же среде, в которой проводится и мануальное тестирование. Автоматизация тестирования нуждается в данных – они требуются при каждом запуске тестирования; и помните, такие запуски происходят намного чаще, чем при мануальном тестировании. Поэтому у вас не получится с легкостью обновлять все тестовые данные в любой момент, чтобы запустить скрипты автоматизации, пока ручное тестирование достигнет логической точки обновления. Такое положение дел нежелательно по ряду причин; вам вместо этого нужно быть готовым к запуску автоматизированного тестирования в любой момент. К счастью, существует несколько различных стратегий, которые вы можете использовать в подобной ситуации (и, вероятнее всего, вы будете их комбинировать):

- по завершении тестирования возвращайте данным их прежнее состояние;
- создавайте данные в рамках процесса выполнения тестирования;
- определяйте частичные наборы данных во всех затронутых приложениях, которые вы можете каждый раз безопасно менять;
- накапливайте базу наборов данных, передаваемых в автоматизацию, пока не будет достигнута следующая логическая точка обновления.

Принимайте во внимание всю систему, а не одно приложение

Организация автоматизированного тестирования часто превращается в упражнение на координацию взаимодействия, так как в рамках тестирования бизнес-процессы нередко затрагивают несколько приложений. Если многие системы опираются на некоторые шаги, выполняемые вручную, то ваша автоматизация будет зависеть от того, как вы скоординируете взаимодействие с ними. Создавая автоматизацию лишь для одной системы, вы рискуете зайти в тупик, когда понадобится, чтобы ваше решение могло взаимодействовать с другими. К тому же *инструменты для изолированного автоматизированного тестирования* могут не сработаться вместе, поэтому сначала подумайте о вашей системе приложений и о бизнес-процессах в целом, прежде чем активно инвестировать в одно специфичное решение для одного приложения.

Звучит как призыв «Не следуйте пирамиде автоматизации тестирования»

Автоматизация тестирования должна следовать принципам пирамиды (рис. 7.2). Большинство тестов должны представлять собой модульные тесты, быстро выполняемые на уровне компонента. Самая тяжелая работа сосредоточена в сервисе или на функциональном уровне, где мы тестируем через API и лишь некоторые тесты выполняются из пользовательского интерфейса (UI). Когда мы говорим о ручном тестировании, то обычно подразумеваем работу через UI. Многие подходы к автоматизации тестирования пытаются имитировать поведение ручного тестировщика, когда автоматизируются тест-кейсы, выполнявшиеся до этого из UI. Но горькая правда в том, что UI слишком медлительный и ненадежный. Вам нужно перепроектировать ваш подход к тестированию таким образом, чтобы извлечь пользу из уровня сервиса; иначе вам придется терпеть увеличение количества поправок, вносимых в скрипты автоматизации.

Автоматизация тестирования для устаревших приложений – довольно неоднозначное занятие. Вам необходимо покрыть большую долю функциональности, и дело это неэкономное, если охватывать все сразу. Я советую начинать с небольшого количества регрессионных тестов для устаревших приложений, чтобы вы могли убедиться в качестве приложения, прежде чем переходить к приоритетной функциональности. Затем, основываясь на изменениях, внесенных в приложение, или на областях, для которых вы осуществляете перепроектирование, создайте дополнительные тест-кейсы. При добавлении новой функциональности создавайте новые тест-кейсы на основе нововведений в коде. В случае перепроектирования приложения сначала создайте тест-кейс,

убедитесь, что он работает и проходит проверку в текущей версии приложения, а затем, после перепроектирования, убедитесь, что все работает[1].

Рис. 7.2. Пирамида автоматизации тестирования. Чем медлительнее уровень, тем меньше мы будем его использовать при автоматизации

Управление качеством и показатели качества

Как я уже говорил, вам придется найти общие способы измерения показателей для формирования выводов о качестве и о процессе его обеспечения. Измеряйте, как хорошо ваш процесс работает с автоматическим выявлением проблем, связанных с качеством, и насколько быстро и точно это происходит. Работа зачастую останавливается из-за неудовлетворительной поддержки автоматизации; показатели помогут вам следить за этим. Обращайте внимание на такие показатели, как длительность проведения вашей регрессии и количество ложноположительных результатов автоматизированного тестирования. Показатель качества результатов должен быть основан на инцидентах среды эксплуатации и, возможно, на дефектах, обнаруженных на фазе стабилизации. Не измеряйте дефекты, которые команда поставки обнаружила самостоятельно: задача команды – обнаружить как можно больше дефектов. Документирование и измерение дефектов, которые находит команда поставки, здесь принесет не очень много пользы; позвольте команде сосредоточиться на выявлении и передаче. Только в том случае, если дефекты перейдут от коман-

[1] На форуме DevOps Enterprise в 2015 году группа создала руководство по автоматизации тестирования для устаревших приложений. Рекомендую изучить работу *Tactics for Implementing Test Automation for Legacy Code*, если вам нужно больше подробностей [3].

ды поставки в другие фазы, вам понадобится документация, чтобы управлять их передачей от команды к команде.

Для того чтобы внедрить это в ваши команды, вам пригодится применение дифференцированной терминологии для различения фаз, в которых была обнаружена проблема. Мне нравится использовать слова *баг*, *дефект* и *инцидент*. Баг – это то, что обнаруживают Agile-команды и что не позволит владельцу продукта принять историю. В таком случае не будет составляться формальная документация; вместо этого тестировщик предоставит как можно больше информации для разработчика, чтобы тот исправил проблему. Как только баг будет исправлен, история заново будет проходить проверку, и процесс будет повторяться до тех пор, пока владелец продукта не примет наконец историю. Мы не измеряем баги каким-либо формальным способом. Когда в работе задействуется другая команда, например команда стабилизации, мы называем проблемы дефектами и управляем ими по жизненному циклу дефекта; мы можем измерить, как много дефектов ускользнуло от Agile-команды. Эти дефекты можно использовать для обдумывания того, какие ограничения мешают Agile-команде находить их самостоятельно; также они помогают продумывать ответные меры. В среде эксплуатации мы обозначаем дефекты кодом «инцидент», и это уже тот показатель обеспечения качества, который мы хотим измерить, так как он оказывает влияние на клиентов. Инциденты являются замечательным показателем качества продукта или сервиса.

Также существуют различные показатели работы в среде эксплуатации, например длительность работы и функциональная доступность, но про это мы поговорим в главе 11, посвященной обслуживанию приложения.

Первые шаги вашей организации

Оценка процесса обеспечения качества в вашей организации

Эта деятельность в некотором роде подобна процессу систематизации потока ценностей из главы 1. Здесь мы опять же должны приготовить доску или еще как-нибудь использовать настенное пространство, а также запастись офисным пластилином и картами системы.

Сначала воссоздайте высокоуровневый рабочий процесс на доске, демонстрируя требования к процессам изготовления, включая все связанные с этим шаги, представленные на картах. Затем возьмите карты другого цвета или ручку, перечислите все виды деятельности для обеспечения качества и покажите, где находится их место в жизненном цикле разработки.

Для дополнения картины спросите себя, все ли области вы покрыли: производительность, безопасность, доступность и т. д. Ничего страшного, если у вас появятся области, которые не были включены в список; просто расскажите о них между делом.

Теперь вашей команде необходимо подумать об автоматизации: что можно автоматизировать, а следовательно, проверять намного чаще? По-

думайте о том, как разделить виды деятельности на автоматизированную часть, которую можно выполнять раньше и чаще, и часть, которую все еще нужно выполнять вручную. Помните, что автоматизированные действия не потребуют больших усилий, поэтому частое их выполнение не повлечет за собой повышения стоимости проверки.

Теперь вам необходимо определить возможности, чтобы проводить проверки качества как можно раньше; это исследование нужно выполнять вручную. По каждому виду деятельности составьте возможные пути, для того чтобы как можно раньше обеспечить полное или хотя бы частичное покрытие.

И наконец, создайте бэклог, с помощью которого ваша команда сможет перейти к обеспечению качества, а затем начните переход.

Оценка качества

Как обсуждалось в этой главе, измерять качество не так-то легко. Многие показатели актуальны только на протяжении определенного времени, когда вы обращаетесь со специфичными областями. Соберитесь с вашими лидами по качеству и тестированию и на листе бумаги перечислите все метрики и показатели, которые вы используете для оценки качества. Затем определите, какие из них являются объективными и автоматизированными.

Если у вас получится хороший набор автоматизированных и объективных метрик, проработайте его вместе, чтобы сократить этот список. Я думаю, что здесь вполне естественно представить себе два показателя: длительность успешного выполнения регрессионного тестирования и инциденты, выявленные в среде эксплуатации за определенный период. Эти два показателя достаточно непротиворечивы и применимы во всякого рода компаниях, но у вас все еще остается необходимость определить несколько дополнительных метрик, связанных со стороной бизнеса.

ГЛАВА 8

Управляйте людьми, а не «ресурсами»

> Управлять командами было бы легко, если бы в них не было людей.
>
> *Анонимный автор*
> *(мой первый куратор)*

Некоторые болезненно реагируют на то, что людей называют ресурсом и применяют в их отношении термин «управление ресурсами». И я отчасти с этим согласен. Ресурс – нечто, чем можно распоряжаться без индивидуального подхода; вы можете обращаться с мешком песка точно так же, как с другим мешком песка, когда речь идет о постройке плотины. С людьми это не пройдет. Наверняка, работая над проектами, вы создавали план, который включает в себя некоторое количество неопределенных штатных единиц (full-time equivalents, FTE). Позднее вы начинаете подставлять имена и понимаете, что вам нужно больше или меньше ресурса, исходя из оценки работы людей, которые заняты в данном проекте. Один Java-разработчик не похож на другого; и если честно, ни один из нас не обрадовался бы, узнав, что к нему относятся как к ресурсу: мол, не можешь работать – заменим тебя кем-то, кто справится не хуже! Может быть, мы лишь воображаем, что можем управлять людьми как безликими «штатными единицами»?

Пожалуй, ранее на производстве заменить рабочего было намного легче, чем Java-разработчика в современной компании. В этой главе я расскажу о том, как

гуманно осуществлять управление людьми[1]. Ценность этих советов в том, что они применимы на всех иерархических уровнях, от менеджеров низшего звена до CIO. В конце концов, от того, как руководители относятся к управлению сотрудниками, зависит культура в организации. Если все сделать правильно, подать людям пример и вдохновить их, вы почувствуете разницу.

Когда я после окончания университета только-только начал работать, я наблюдал в своей организации кое-что очень непривлекательное. Мой руководитель, человек открытый и с широким кругозором, выслушав меня, дал мне один из важнейших советов в моей жизни: «Мирко, ты не сможешь изменить всю организацию, но ты можешь изменить свой участок работы. Если ты менеджер, управляй своими командами так, как хотел бы, чтобы обходились с тобой, а затем пожинай результаты. По мере карьерного роста твое влияние будет все заметнее и за тобой будут подтягиваться остальные. Будет нелегко, но кто его знает… однажды ты можешь оказаться на самой вершине и в том, что корпоративная культура где-то провисает, будешь винить только себя» [1]. Сегодня я все еще далек от карьерных вершин, но это наставление не забываю и по сей день.

В этой главе я расскажу, как лично я пытаюсь управлять. Те, кто работал или все еще работает на меня, знают, что я не идеален, но при любой возможности стараюсь применять данные практики. Оттачивать навыки управления людьми можно всю жизнь.

Личные встречи

Первая из практик, которую я вам настоятельно рекомендую (если вы еще к ней не прибегали), – это регулярное проведение переговоров с глазу на глаз. Довольно тяжело управлять человеком, который представляет собой некую виртуальную фигуру, о которой вы можете судить лишь по продукту, над которым этот сотрудник работает. И нет, открытая политика – не то же самое, что проведение личных встреч! При открытой политике ваши собеседники могут почувствовать, что вы проводите встречи из принуждения, и поэтому встреча может оказаться менее результативной, чем вам хотелось бы. Тот факт, что вы предпочли пообщаться один на один, говорит людям, что вы не жалеете для них свое время.

Получасовых встреч раз в неделю или две вполне достаточно. Вам нужно пользоваться этой возможностью, чтобы со временем больше узнать о человеке, не чувствуя неловкости от того, что вы вторгаетесь в его личную сферу. Всегда давайте собеседнику возможность рассказать или спросить о том, что его

---

[1] Если вы слышали о Марке Хорстмане и Майкле Озанне из Manager-Tools.com, то легко убедитесь, что значительная доля материала для этой главы написана под впечатлением от их работы. Их подкасты и конференции (которые они сами называют тренингами) – одни из лучших руководств, которые там есть. Подкасты бесплатные, а конференции стоят недорого и доступны по всему миру. Очень советую послушать!

волнует, а затем поделитесь своими мыслями и информацией, которая может оказаться для него важной. После проведения ряда личных встреч вы обнаружите, что они становятся все более содержательными и к тому же вам все реже предлагают проводить внеплановые совещания. Общение с глазу на глаз – не только шанс получше узнать человека; для вас это шаг к тому, чтобы развить свои управленческие навыки и больше сосредоточиться на своей работе. А все потому, что те, кто на вас работает, больше не будут беспокоить вас каждую неделю по несрочным вопросам, если у них будет возможность регулярно совещаться с вами.

По моему опыту, нельзя откладывать более четверти (25 %) личных встреч, чтобы они воспринимались как значимые и положительно влияли на ваши отношения с сотрудниками и на их работу. Все, кого я знаю, через непродолжительное время полюбили такой формат работы, так что попрактикуйте такие мини-совещания пару месяцев, прежде чем сделать вывод, нужны ли они вам[1]. Многие люди, с которыми я проводил встречи, потом сами затевали нечто подобное в своих командах. А значит, дело стоит того.

Обратная связь

Все, кого я встречал, желали получать больше отзывов от своих руководителей, в то время как руководители не очень-то горели желанием предоставлять обратную связь (особенно если речь идет о критической оценке, которую надо тактично преподнести; с положительной оценкой как-то попроще). Если вдуматься, конструктивная критика может быть очень полезна, но преподносить ее нелегко. К тому же есть удачные и неудачные способы изложить ваше мнение. Снова возвращаясь к «совершенствованию, автономии и целям» Дэна Пинка [2], хочу сказать, что вам нужно поощрить сотрудника к самосовершенствованию, придерживаясь всех трех аспектов.

Прежде всего сосредоточьтесь на конкретных ситуациях – не допускайте обобщений типа «вечно ты опаздываешь» или «ваш рабочий продукт оставляет желать лучшего». Объясните, что именно в поведении сотрудника вас не устраивает, и попросите изменить отношение к делу. Например: «Когда ты заранее не докладываешь, о чем пойдет речь на совещании, как произошло сегодня с поставщиком, мы тратим много времени на пустую болтовню и в результате упускаем главное. Не мог бы ты учесть это на будущее?»

Как вы видите, в этом примере оценивается недавнее событие (совещание, проведенное утром того же дня), объясняется, чего вы хотите (таким образом вы учитесь четко формулировать запросы!), а сотруднику предоставляется возможность самому решить, как исправиться в дальнейшем. Конечно же, со-

[1] На сайте Manager Tools есть много материала о личных встречах, включая вводные материалы и ряд подкастов на тему управления такими встречами. Если вы столкнетесь с особенными трудностями, опять же загляните в Manager Tools: там есть много советов о том, как использовать личные встречи в особых обстоятельствах.

трудник может попросить вас конкретизировать ваши пожелания, но эта простая модель обратной связи прекрасно работает с мотиваторами Дэна Пинка.

Я пользуюсь следующим примером, который помогает мне сосредоточиться на ключевых элементах отзыва (поведение, последствия и необходимость перемены): когда ты делаешь *x*, происходит *y*, что для нас не вполне приемлемо. Не мог бы ты действовать по другой схеме в следующий раз?

Вы можете использовать подобный пример и для положительной оценки. В таком случае вы не предлагаете перемены, а просите продолжать в том же духе. Такой тип положительного отзыва в большей степени соответствует трем мотиваторам, чем просто «молодец» или «ты хорошо справился».

Делегирование обязанностей

В самом начале моей менеджерской карьеры меня смущала мысль о передаче другим сотрудникам той работы, которую я мог сделать сам или которая не вызывала энтузиазма (например, заполнение документов). Послушав подкаст *Manager Tools*, я узнал о концепции под названием «Экономика управления 101» [3], которая подразумевает, что если за счет делегирования задачи можно выполнить ее с меньшими затратами, то не передать ее будет неэкономично. Эта концепция изменила мой подход к управлению: я больше не комплексовал, передоверяя задания тем, кто на меня работает. С другой стороны, я вспоминал, как в свое время напрягалась моя начальница, стесняясь переложить часть работы на меня. Когда я сам предложил ей помощь, она очень обрадовалась. Если вы примете это к сведению, то со временем сможете пользоваться большим доверием своего руководителя.

Чтобы сотрудник оценил ваше доверие, используйте три мотиватора. Объясните, как возлагаемые на сотрудника обязательства помогут научиться чему-то новому, как он разгрузит вас, и наконец дайте сотруднику самому определить, как достичь ожидаемого результата.

Например: «Майкл, ты не мог бы мне помочь с еженедельным отчетом? Если ты сможешь сделать его вместо меня, я успею хорошенько подготовиться к предстоящему совещанию и доложить о ходе работ по проекту заинтересованным лицам. Отчеты необходимо составить по нашему стандартному образцу. Я тебе с удовольствием покажу, как я это делал, но если ты найдешь способ лучше, действуй по своему усмотрению. Первые отчеты можем для удобства составить вместе. Если есть вопросы, давай обсудим».

Создание «культуры без обвинений»

Будучи руководителем, вы должны защищать участников команды, когда у кого-то возникают претензии к ним. Если в команде появилась проблема, вам придется разбираться с ней, не пытаясь найти и наказать виноватого. И команда это оценит. А вот если хотите отметить хорошую работу, не скупи-

тесь на похвалы в адрес конкретного человека: чем больше вы хвалите представителей вашей команды перед всем коллективом, тем больше вас ценят. Такое поведение поощряет людей выполнять свою работу наилучшим образом. И в организации в целом вы приобретете хорошую репутацию, если команда работает успешно. Чрезмерная суровость и нежелание отмечать выдающиеся достижения – весьма неэффективная тактика.

В итоге вы формируете в команде так называемую «культуру без обвинений». Если вы вспомните мое замечание о системе, которая влияет на поведение людей, то становится ясно, что в любой ошибке прежде всего стоит винить систему, а не отдельного сотрудника. В Etsy существует практика, которая состоит в том, что новому инженеру позволяют проводить развертывание в среде эксплуатации прямо в первый день работы. Если новичок сможет испортить что-то в среде эксплуатации, это значит, что система чересчур уязвима и не справляется с простыми ошибками [4]. Поэтому, какие бы проблемы вас ни ожидали, ваше дело – не выискивать, кто виноват, а пытаться понять, как нужно изменить систему, чтобы на будущее избежать повторения старых ошибок. Если выстраивать модель поведения для реагирования на инциденты и предотвращать взаимные обвинения, то в вашей команде наметится сдвиг к более позитивной и воодушевляющей культуре, которая поощрит сотрудников добиваться наилучших результатов для организации. Но как вы можете измерять эту культуру, чтобы наглядно видеть улучшения?

Оценка культуры вашей организации

Существует несколько способов измерения показателей культуры. Один из них мы раскрыли в работе по DevOps-метрике «Измерение производительности, эффективности и культуры для оптимизации DevOps-трансформации», в исследовательских средствах Westrum:

- в моей команде ведется активный поиск информации;
- в моей команде проблемы рассматриваются как возможность роста, а тех, кто сообщает о проблеме, не наказывают;
- в моей команде разделяют ответственность;
- в моей команде поощряется и вознаграждается кросс-функциональное сотрудничество;
- в моей команде провал служит стимулом к исследованию проблемы;
- в моей команде приветствуются новые идеи.

Лично я предпочитаю использовать слегка упрощенную версию внутреннего показателя лояльности потребителей (Net Promoter Scope, NPS), который был заимствован из общего NPS, используемого для измерения удовлетворенности клиентов. Я использую следующие четыре утверждения в одном из своих проектов, в котором каждый член команды должен оценить, насколько они соответствуют реальности:

- я порекомендовал бы команду друзьям, так как это хорошее место работы;
- у меня есть инструменты и ресурсы для надлежащего выполнения моих обязанностей;
- я редко думаю об уходе из команды или из компании;
- моя роль в проекте дает мне возможность полноценно использовать свои навыки и способности.

Вы можете использовать любые из этих утверждений, при желании добавляя другие, но важно, чтобы вы не искали абсолютную ценность: ваша задача – постоянно отслеживать тенденции. Думайте, улучшаете ли вы положение в вашей части организации. Запомните совет моего старого руководителя: вы можете контролировать лишь свой фронт работ. Следовательно, если ваша команда непривлекательна для потенциального соискателя, то это ваша вина. Четко оценивать культуру команды очень важно, иначе вы не сможете понять, улучшается ситуация или нет.

В список занятий в этой главе я добавил одно, касающееся измерения того, насколько хорошо менеджеры управляются со своими командами. Я в шоке от того, что почти всегда, задавая директорам вопрос, как они анализируют качество управления людьми в их командах, я слышу, что они об этом и не помышляют. Неудивительно, что потом они встают перед фактом, что управление не улучшается и смена культуры замедляется! Не совершайте ту же ошибку, внесите измерение показателей культуры в список ключевых показателей производительности (Key Performance Indicators, KPI). Пользуйтесь общими средствами измерения, приведенными здесь, в качестве показателя KPI и передайте его в пользование вашим менеджерам.

В этой главе я рассказал, как управление ролями меняется, когда вы отстраняетесь от устаревшей модели управления, и предложил вам некоторые инструменты управления работниками умственного труда. Помните, что мы не работаем на фабрике со сборочной линией, мы имеем дело с интеллектуальной ценностью и креативными начинаниями! А кроме того, мы работаем с технологией, поэтому настало время перейти к последней части книги и обсудить технологические аспекты трансформации.

Первые шаги вашей организации

Проводите личные встречи

Очень важно выкраивать в рабочем расписании время для каждого из ваших сотрудников – для начала не реже, чем раз в две недели, а затем иметь возможность проводить регулярные еженедельные встречи. Пусть они длятся по 30 минут: сначала 15 минут для сотрудника, а затем 15 минут для вас. Зачастую сотрудники превышают ограничение в 15 минут, и это не страшно. Вы всегда сможете найти время на неделе, для того чтобы вы-

сказать то, что не успели сообщить в прошлый раз. Я также настоятельно рекомендую записывать важные моменты и отсматривать записи с прошлой недели, чтобы собрать объемную информацию, когда дело дойдет до рассуждений о производительности, которые предоставят вам показатели для вашего отчета о сотруднике.

Составьте список показателей KPI для ваших менеджеров

Вы, вероятно, слышали поговорку «что измеряешь, то и получаешь» (you get what you measure). И хотя культура с трудом поддается замерам, кое-что можно сделать и здесь:

- пользуйтесь внутренними показателями NPS, о которых я говорил в этой главе, и приспосабливайте их для работы с командой в качестве высокоуровневого показателя;
- измеряйте эффективность личных встреч, которые проводят ваши менеджеры. Это позволит вам точнее понять, эффективные ли взаимоотношения выстраивает ваш менеджер;
- точечно проверяйте прочность отношений ваших менеджеров. Ненавязчиво интересуйтесь такими деталями, касающимися сотрудников, о которых руководитель должен знать, чтобы понимать, положительно ли развиваются отношения. Например, знает ли он имена детей и партнеров своих подчиненных? А что может сказать об их увлечениях?

И да, конечно же, вы должны начать с себя, чтобы ни один менеджер не мог упрекнуть вас в том, что вы видите соломинку в чужом глазу, не замечая бревна в своем.

Заключение части Б

Мы подошли к завершению второй части книги, рассказывающей о людях и организации процессов в современных ИТ-организациях. Мы рассмотрели различные способы обеспечения ваших сотрудников организационной структурой, бизнес-контекстом и необходимой обратной связью, чтобы добиться применения трех мотиваторов, описанных Дэном Пинком: автономии, совершенствования и цели. Мы потратили дополнительное время на погружение в изменения, которым подвергаются организация и люди и с которыми сражаются в процессе трансформации; переход от тестирования к обеспечению качества, где нашли пример того, как укрепить автономию, совершенствование и цели посредством организации и проектирования процессов. Успех или провал трансформации будет напрямую зависеть от ваших сотрудников, поэтому уделите им достаточно времени. В следующей части мы поговорим о технической стороне. Технологии сегодня находятся в самой середине всех бизнес-идей, поэтому нам нужно управиться с ними, чтобы добиться успеха.

ЧАСТЬ B

Технологические и архитектурные аспекты

За последние годы мне довелось поучаствовать в дискуссиях о DevOps со многими организациями. Обычно в центре обсуждения были инструменты, практики и культура. Вы поймете, почему это так, если почитаете популярные материалы по DevOps. Если размышлять о том, что может повлиять на время вывода продукта на рынок, то мне думается, что есть место для небольшой хитрости: архитектура играет огромную роль в том, какой конечный результат вы получите после внедрения DevOps. Хотя архитектуру тяжелее менять и о ней не настолько интересно разговаривать. Ориентироваться на автоматизацию и развивать культуру – это правильно. Но есть и третий элемент, о котором мало кто говорит, – архитектура трансформации. Если вы работаете с большими монолитными legacy-приложениями, то вскоре достигнете предела возможностей. Поэтому инструменты, методы и процессы также играют важную роль.

В третьей части этой книги будет рассказываться про то, что вряд ли будет долгое время сохранять актуальность. Рекомендации из первых двух частей вы сможете применять на практике еще долго. Материал третьей части будет устаревать быстрее. Идеи развивались с течением времени, их становилось все легче использовать с появлением новых инструментов и технологий. Мне не терпится увидеть, как все это будет развиваться дальше. К примеру, бессерверные архитектуры, такие как Amazon Lambda, еще только маячат на горизонте, а через пару лет вынудят меня переписать всю третью часть этой книги. Однако я убежден в том, что основные идеи, изложенные ниже, выдержат проверку временем. Поэтому я добавил в книгу раздел о технологиях, отчетливо осознавая, что многое из сказанного мной скоро утратит значимость.

В этой части книги мы рассмотрим:

- правильные способы выбора DevOps-инструментов;
- какую DevOps-архитектуру можно считать хорошей;

- как развивать архитектуру вашего приложения;
- как использовать связи между DevOps и облачными вычислениями;
- как выглядит каждая из моделей доставки ИТ-продукта и как эти модели могут повлиять на вашу организацию.

ГЛАВА 9

Различные модели доставки продукта

> Господа, мы собираемся неустанно преследовать идеал, заведомо зная, что нам его не достичь, так как идеального не существует. Но мы будем его преследовать, так как в процессе его преследования мы сможем достичь совершенства.
>
> *Винс Ломбарди,*
> *цитата из «Игры всей моей жизни»*
> *Чака Карлсона*

В этой главе я опишу три различные модели, которые на данный момент используются для успешной доставки ИТ-продукта. Также речь пойдет о том, что заставит эти модели работать. Но, как я уже несколько раз упоминал, задача выходит за рамки одного только технического совершенствования. Она зависит от организации – провести грамотные преобразования в самой организации, чтобы обеспечить поддержку модели доставки.

Обзор моделей поставки продукта

Я встречал три модели, которыми активно пользовались или к которым стремились в крупных организациях. Они позволяли иметь дело с устаревшими приложениями и с современными цифровыми технологиями одновременно.

1. Непрерывная поставка (такой тип описан Джезом Хамблом и Дэвидом Фарли в книге о непрерывной поставке) позволяет вам автоматически развертывать приложения во всех средах, начиная со среды разработки и заканчи-

вая средой эксплуатации, а также автоматически тестировать их. Эта модель основана на постоянной среде для развертывания.
2. Облачная доставка: популяризована Netflix. Согласно этой модели, каждый раз создается новая среда, а старая уничтожается тогда, когда новая докажет свою работоспособность. Это модель, которая подразумевает отсутствие потерь времени при развертывании.
3. Доставка с поддержкой контейнеров: получила распространение с увеличением популярности Docker как технологии контейнеризации, поддерживающей микросервисную архитектуру.

Существует и четвертая модель, которая на данный момент еще только начала развиваться. Она основана на бессерверных технологиях, таких как Amazon Lambda. На момент написания я еще не успел поработать с клиентами над тем, чтобы выстроить модель доставки для бессерверных технологий, и даже не сформировал таковой в принципе. Возможно, в следующем издании я уже смогу дополнить главу описанием четвертой модели доставки.

Вероятнее всего, вы будете заинтересованы в ускоренной доставке продукта, поэтому будете использовать комбинацию вышеназванных моделей. Когда я буду говорить об этих моделях в контексте трансформации, нужно иметь в виду, что они не применяются ко всему и сразу, но внедряются постепенно, по мере того как вы преобразуете свои приложения и технологии в соответствии с этими моделями доставки.

Модель A: непрерывная доставка

Эта модель, вероятно, самая известная, и она существует уже довольно продолжительное время, хотя многие компании пока только пытаются ее достойно реализовать. Непрерывная доставка подразумевает, что вы потенциально можете развертывать в продуктивной среде каждую сборку, благо все необходимые для этого шаги автоматизированы. Слово «потенциально» нужно понимать так же, как его употребляют в мире Agile: оно означает, что важность состоит в возможности осуществить это, а не в фактическом осуществлении. Вы также легко можете предпочесть не развертывать это автоматически в продуктивную среду – например, прибегнуть к ручному тестированию или дальнейшему совершенствованию системы. Можно использовать эту модель доставки и для облачных, и для локальных окружений.

Самый распространенный шаблон среды для развертывания – это постоянные среды (например, среды, в которых не один раз разворачивалось ПО). Шаблон часто необходим для работы с legacy-приложениями, для которых требуется специфичная конфигурация среды. Вот почему применение этой модели здесь прекрасно подходит. Преимущество замены модели, подразумевающей ручную работу, данной моделью состоит в том, что увеличится скорость доставки, уменьшится риск для вашего процесса доставки за счет того, что больше не будет шагов, осуществляемых вручную, увеличится частота прове-

рок и получения обратной связи, а также сократится количество бесполезного шума в процессе доставки благодаря достижению прозрачности во всем жизненном цикле доставки продукта.

Описание возможностей

Для успешной работы с непрерывной доставкой вам необходимы четыре возможности. Я лишь кратко опишу основы, а остальное вы при необходимости почерпнете из других источников, коих несметное множество.

1. Создание приложения – одна из основных возможностей, следствие которой: нет приложения – нет и всей остальной деятельности. Она подразумевает управление исходным кодом, обеспечение качества со стороны разработки (статический анализ кода, дружеские проверки, unit-тесты), управление рабочими процессами разработки (возможность отслеживания всех изменений, внесенных требованиями к этим процессам), процессы компиляции, сборки и упаковки, а также менеджмент пакетов. Освоение данной возможности крайне важно для реализации всего остального. И хотя существует множество успешных шаблонов, которые можно применять в отношении разных технологий (например, пользоваться возможностями Google или качать конфигурацию Jenkins для сборки Java-приложений), ваши возможности продолжат расти и будут меняться вместе с контекстом каждой из используемых вами технологий.
   В идеальном случае вы со временем добьетесь непрерывной интеграции (CI), когда ваши приложения собираются по принципу прохождения последовательности шагов, чем вы можете воспользоваться в работе со всеми своими приложениями и технологиями; но зачастую вы будете встречаться с технологиями, для которых это может оказаться нецелесообразным. Например, когда мы работали с Siebel, компиляция занимала более двух часов, что уже слишком долго для CI. Тем не менее автоматизация этой возможности дается легче, чем все остальное. Частота компиляций может отличаться, но главное – преследовать цель уменьшения ручного вмешательства в создание программного пакета из кода, переданного системой управления конфигурациями (SCM). Я занимался автоматизацией для Siebel, мейнфреймов и многих других технологий. И это не всегда давалось легко, но было возможно.
2. Развертывание приложения – осуществляется уже немного сложнее. В целом это означает, что мы будем брать программный пакет из системы управления пакетами и разворачивать его в существующей среде. Подобная тактика прекрасно подходит и для полной автоматизации. Чтобы ее реализовать, вам стоит кое о чем подумать.

   - Осведомленность о среде: для успешного развертывания вам нужно знать, *куда* вы будете разворачивать приложение. Весьма вероятно, что у вас будет топология сред с различиями на разных уровнях (мульти-

арендность в низших уровнях и избыточное количество компонентов ближе к выводу в продуктивную среду), поэтому необходимо знать, на каком сервере будут разворачиваться ваши компоненты. Также во время развертывания необходимо знать специфичные для сред настройки (в частности, IP-адреса, имена серверов, значения, присвоенные соединениям) и, в случае инкрементальных развертываний, последнюю развернутую версию.

- Сохранение независимости среды программных пакетов: поскольку в процессе создания пакета вы не знаете, где он будет разворачиваться, все специфичные для среды настройки нужно абстрагировать. Вы можете осуществить это в виде конфигурационного файла или при помощи переменных, которые используются либо во время развертывания, либо во время исполнения.
- Полное или инкрементальное развертывание: при полном развертывании будет произведена полная замена приложения в окружении, а следовательно, вам ни о чем не придется беспокоиться. Сначала вы удаля-

Модель А – непрерывная доставка

Согласно модели А, приложения автоматически развертываются в постоянную среду (облачную или локальную)

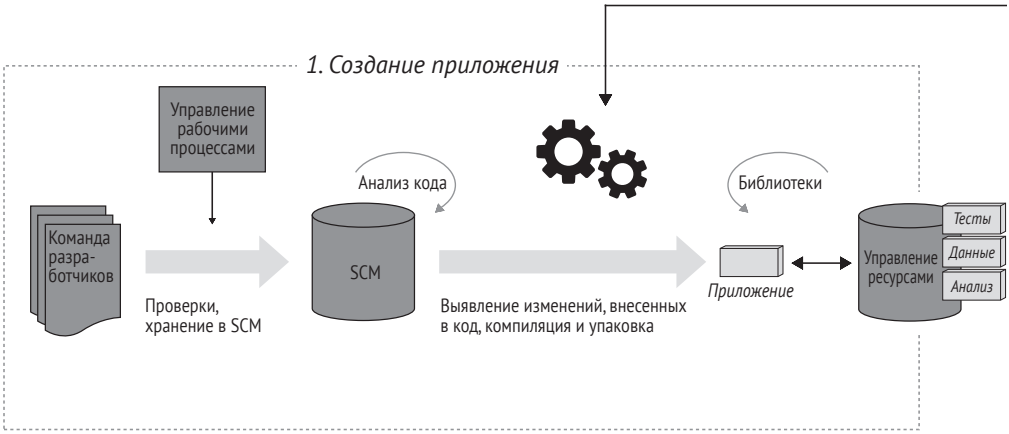


Рис. 9.1. Модель А – непрерывная доставка:
при непрерывной доставке автоматизация позволяет добиться постоянной среды

ете все, что связано с предыдущей версией (после создания резервной копии – на случай, если развертывание не будет удачным), а затем развертываете новую версию в окружении. Это полное развертывание автоматизировать намного легче, так как при этом будет осуществляться один и тот же процесс. Инкрементальные развертывания осуществляются быстрее, но потребуют применения более сложных процессов для оптимизации; именно поэтому организации зачастую начинают с полного развертывания, если это поддерживается технологией (например, структурные изменения для транзакционных таблиц всегда осуществляются инкрементально). Для выполнения инкрементального развертывания важно знать последовательность инкрементальных пакетов и какая версия сейчас находится в среде, чтобы можно было правильно определить ряд файлов, которые необходимо изменить. Хорошая практика – автоматизированная проверка того, что среда находится в ожидаемом состоянии, перед началом развертывания: риск того, что что-нибудь пойдет не так, при инкрементальных развертываниях намного выше. Также существует большой риск конфигурационного сдвига из-за изменений, осуществляемых вручную, или неудачных попыток развертывания по причине того, что окружение не было подготовлено перед самим развертыванием.

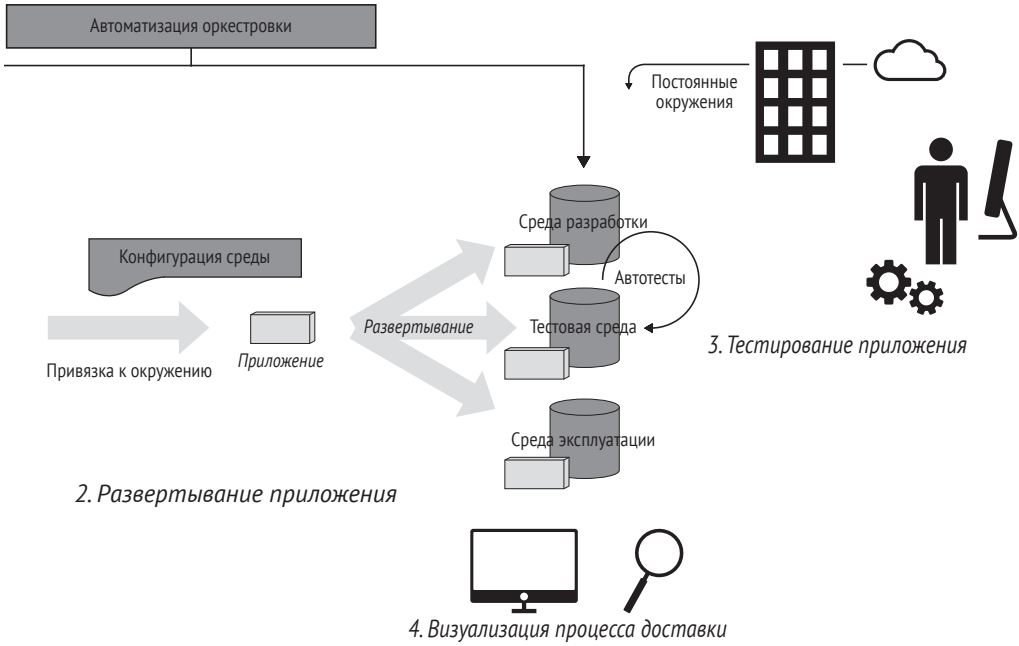


Рис. 9.1 (окончание)

3. Тестирование приложения: нужно автоматизировать как можно большую долю процесса тестирования и отслеживать покрытие тестирования, которое может отличаться от среды к среде. Различные уровни тестирования относятся к данной возможности: тестирование приложения, интеграционные тесты, тестирование производительности, тестирование безопасности, тестирование готовности к эксплуатации, а также все то, что можно автоматизировать. По большому счету, все ручное тестирование и руководства по управлению ручным тестированием также относятся к этому пункту.
4. Визуализация процесса доставки: мне кажется, невозможно улучшить то, чего вы не видите. Модель доставки в целом не является чем-то, что можно просто реализовать, – и все тут, вам придется продолжить ее настраивать и улучшать. По моему опыту, первая реализация занимает от трех до шести месяцев; затем реализация достаточно сильно изменяется в течение следующих шести–двенадцати месяцев, по мере того как вы развиваетесь. Чтобы все шло гладко, вам нужно разработать способ визуализации всего процесса и измерения сообразности и быстроты выполняемой работы. Раньше это осуществлялось посредством файлов или Excel-таблиц, как и многие другие процессы в ИТ, но сегодня уже существуют новые инструменты визуализации и открытые программные решения, которые облегчают использование этой возможности. Capital One даже решили выложить свое решение для создания панели наблюдения, соответствующей методологии DevOps, в открытый доступ, и это решение успешно переняли другие организации, что помогло им, в свою очередь, справляться с DevOps-трансформацией [3]. Это не самая сложная из всех возможностей в этой модели доставки, но ее сильно недооценивают. Зачастую организации не уделяют достаточно времени и сил для реализации этой возможности.

Аспекты трансформации и воздействие на организацию

По мере того как вы переходите к непрерывной доставке, вы все чаще задумываетесь о некоторых вещах. Во-первых, управлять конфигурациями крайне важно, без этого вы в действительности не сможете делать все остальное. Управление конфигурациями позволяет вам быстро справляться с эксплуатацией. Весь код (включая тесты и код автоматизации) необходимо содержать в системе управления конфигурациями, чтобы его было легче оценивать и беспроблемно использовать. Переход к этой модели потребует тесного сотрудничества ваших команд по эксплуатации и инфраструктуре с командой платформы, чтобы можно было реализовать *абстрактную конфигурацию среды* (согласно этой практике в конфигурационных файлах вместо конкретных значений назначаются переменные, к которым значения подставляются во время развертывания, тогда, когда значения становятся известными). И для целей автоматизации вам нужно будет задать верные правила доступа к среде. Для команд администраторов это будет казаться потерей контроля над средой, но

если вы осторожно будете управляться с этим процессом, вовлекая все группы в процесс управления изменениями, то переход пройдет намного легче.

Управление изменениями – также важный аспект для перехода к этой модели доставки, и поначалу я недооценивал важность данного занятия (обучение сотрудников, способствование изменениям в организации, обсуждение изменений и положительных результатов, изменение процесса и описание существующих ролей). В конце концов, если у нас получится создать прекрасное решение, то и все остальные воодушевятся, не правда ли? Не совсем, как оказалось. Я заметил эту проблему еще в первой паре проектов и тогда начал принимать ее в расчет – даже в последующих проектах нанимал отдельных сотрудников, которые занимались управлением изменениями. Вышло так, что управление изменениями оказалось весьма необходимой задачей и помогало всей команде. Разработчики не заинтересованы в создании обучающего материала или документировании проекта, а сотрудники, отвечающие за управление изменениями, знают, как сформировать сопроводительный материал, которым наверняка потребуется воспользоваться. Мне кажется, вы можете составить оценку затрачиваемых усилий и стоимости для этой задачи и удвоить показатели данной оценки; вероятно, в завершение работы вам захочется оглянуться назад и подумать о том, что еще можно было бы сделать.

Организационные изменения, преследующие качественные изменения в образе ведения процессов, о которых мы говорили в главе 7, нужно предусматривать в этой модели, иначе у ваших команд доставки, заточенных под быстрое выполнение задач, и отдельной команды по тестированию будут возникать проблемы при взаимодействии.

Кроме прочего, стоит подумать об инфраструктуре для вспомогательной платформы. Очень часто с ней не обходятся так же, как с системой для продуктивной среды. Но подумайте: если ваша продуктивная среда упала из-за дефекта и ваши SCM и инструменты для автоматизации тоже не функционируют, вам придется несладко! Вам нужно иметь продуктивную среду с вашими инструментами, которую вы сможете использовать для осуществления развертываний во все среды (начиная со среды разработки и заканчивая продуктивной средой), и вам потребуется среда разработки с вашими инструментами, чтобы продолжить тестировать, экспериментировать, а также развивать инструменты. Вам не захочется делать все эти вещи со средой, которую вы будете использовать для следующего развертывания продукта.

Модель Б: облачная доставка

Для облачной модели доставки используются некоторые практики, ставшие популярными после распространения концепции непрерывной доставки. Облачные возможности стали более зрелыми, а системы управления конфигурациями сред, такие как Chef, Puppet и Ansible, изменили наше восприятие процесса создания и управления средами. Использование всего этого и отличает данную модель от предыдущей: мы обходимся со средами и инфраструктурой

как с кодом и, таким образом, можем создать любые дополнительные среды легко и просто. Инфраструктура как код означает, что вся инфраструктура задается при помощи конфигурации, которую можно хранить в файле, с которым, в свою очередь, можно обходиться как с исходным кодом программы.

При такой модели мы заново создаем совершенно новую продуктивную среду, в которую помещаем последнюю версию приложения. Затем можно использовать эту среду для тестирования с использованием продукционного трафика, что поможет нам узнать, были ли изменения удачными. В этот момент уже ничто не мешает уничтожить старую продуктивную среду. Это модель доставки с весьма малыми рисками, так как вы можете управлять ими, прибегая к балансированию между тестированием новой среды и моментом перехода на нее.

Описание возможностей

Многие из возможностей схожи друг с другом, но просто используются в разном контексте, в работе с созданием новой среды при каждом развертывании. При этом, например, отпадает необходимость в обслуживании инкрементального развертывания, упомянутого ранее, так как не будет того, что можно развернуть постепенно. Это означает, что вам придется найти другие способы сохранять постоянство между средами (например, как вы передаете

Модель Б – облачная доставка

При модели Б приложения развертываются автоматически после предоставления новой среды из облака или центра обработки данных

Основное отличие от модели А представляют более зрелые практики, применяемые относительно инфраструктуры, – так называемая инфраструктура как код

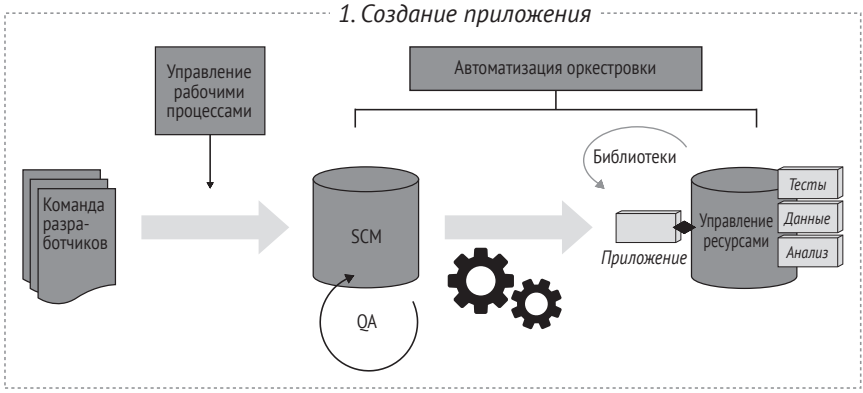


Рис. 9.2. Модель Б – облачная доставка: при каждом развертывании создается новая среда

все транзакционные данные? Или вы храните их в другой неизменной части среды?). Это указывает на ограничения и сложности данной модели доставки, а также на то, почему вы, возможно, не станете использовать ее для всего портфолио ваших приложений.

1. Создание приложения: эта возможность не сильно отличается у разных моделей доставки.
2. Создание окружения: это новая возможность, и, по сути, она подразумевает применение понятия инфраструктуры как код ко всему, что не является кодом приложения, который мы и будем развертывать позднее. Требуемая инфраструктура включает в себя вычислительные среды, сеть, операционную систему и промежуточное программное обеспечение. Потому как нам нужны сведения об инфраструктуре для осуществления процесса развертывания, нам нужно убедиться в том, что вы собрали всю необходимую информацию о конфигурациях. Это очень похоже на то, что вы делаете почти вручную при модели непрерывной доставки для постоянных сред. Здесь же среды постоянно меняются, поэтому необходимо автоматизировать этот процесс. Тут возникает надобность в управлении конфигурациями сред посредством таких инструментов, как Puppet или Chef, в то время как ранее это было необязательно.

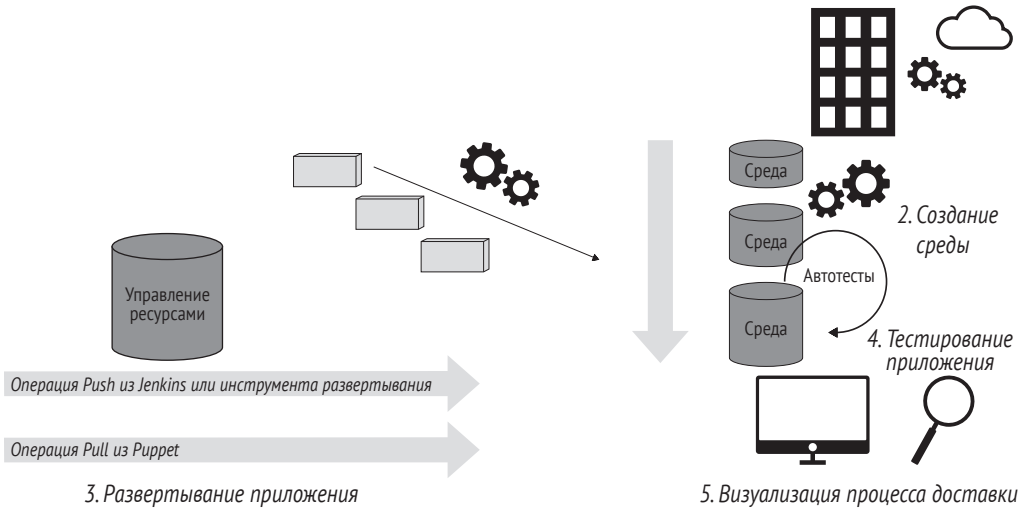


Рис. 9.2 (окончание)

3. Развертывание приложения: при такой модели у нас есть два альтернативных подхода. Мы можем самостоятельно развертывать приложение при каждом создании новой среды или же использовать инструменты для управления конфигурациями сред, чтобы выгрузить необходимую версию приложения. Мне довелось понаблюдать за обеими моделями, и мне кажется, что ваш выбор зависит от ваших предпочтений и контекста. Со временем вы, вероятно, остановитесь на второй модели, потому как она позволяет вам не использовать потенциально дорогостоящие инструменты для развертывания и упростить процесс отладки в целом.
4. Тестирование приложения: эта возможность не сильно отличается от той, что у модели непрерывной доставки, но вы, вероятно, будете выполнять больше тестов, так как инфраструктура часто пересоздается. К тому же набор регрессионных тестов будет больше, так как вам не нужно отключать продуктивную среду для проведения тестов.
5. Визуализация процесса доставки: у вас есть немного больше аспектов, которые нужно визуализировать и оценить, например количество окружений, запущенных на данный момент, и скорость, и безотказность процесса создания новых окружений, но в целом идеи остаются прежними.

Аспекты внедрения и воздействие на организацию

Поскольку инфраструктура не является отдельным объектом для размышлений в контексте платформы, для поддержания этой модели доставки вам придется способствовать взаимодействию ваших команд по обслуживанию инфраструктуры и команд платформы. Все более и более важно, чтобы команды понимали техники автоматизации, нежели разбирались в Windows или UNIX. Вам все еще необходимы эти навыки, но, вместо того чтобы настраивать машины, команда сосредоточивается на концепции инфраструктуры как коде.

Совершенствование навыков в следовании модели непрерывной доставки – первое, что нужно для успешного применения данной модели, так как любые шаги, выполняемые вручную, ослабят преимущества ее использования. В дополнение к этому облачная модель принесет больше выгоды, если вы сможете изменить структуру приложения, для того чтобы можно было пользоваться расширяемостью и гибкостью облака. Мы раскроем эту тему подробнее в главе 12.

Модель B: доставка с поддержкой контейнеров

С ускоренным ростом популярности Docker (который облегчил работу с Linux-контейнерами и превратил работу с контейнерами в господствующую тенденцию) начала развиваться новая модель доставки, которой желают вос-

пользоваться многие организации. Она невероятно хорошо подходит для работы с микросервисной архитектурой благодаря малому объему занимаемой памяти и гибкости контейнеров. Скорость работы такой модели впечатляет, ведь новый контейнер можно создать и развернуть в течение нескольких секунд. В то время как другие модели доставки для этого требуют от семи минут до нескольких часов. Это пока что самая быстрая модель, но при условии, что вы обладаете архитектурой с относительно небольшими контейнерами. (Если вы попытаетесь запустить Siebel или SAP из контейнера, я полагаю, что вы будете разочарованы.) Неизменяемая природа контейнеров (по крайней мере, как она предполагается) поспособствует налаживанию хороших взаимоотношений в организации, так как невозможно вручную изменить контейнеры, после того как они были уже созданы.

Описание возможностей

Как и в случае с предыдущей моделью, здесь возможности наращиваются поверх других, и все возможности, созданные для предыдущих моделей, можно снова и снова использовать; они в некотором роде даже должны присутствовать. В этом случае возможности связаны с созданием и развертыванием контейнеров.

1. Создание приложения: здесь также ничего не поменялось.
2. Создание контейнера приложения: вдобавок к пакету приложения, который хранится в менеджере пакетов, мы создадим еще и контейнеры приложения, которые будут содержать все, что необходимо для запуска автономного приложения.

   Некоторые аспекты автоматизированного предоставления среды относятся к этой возможности, например настройка требуемого хранилища данных, которое должно располагаться в контейнере, а не в среде, когда речь идет о развертывании микросервисов. Некоторые для этой цели используют инструменты конфигурирования среды, но благодаря неизменяемости контейнера вы можете применить более упрощенные подходы для обслуживания этой одноразовой сборки. Управление и обслуживание контейнеров стало новой и важной возможностью.
3. Создание VM/OS-узла. Эта возможность очень похожа на возможность создания среды. Вы создаете очень простую среду, содержащую движок для контейнеров, на который будут развертываться образы.
4. Развертывание контейнера: эта возможность позволяет развертывать контейнеры на узле и включать их (например, прибегая к перенаправлению трафика на этот экземпляр и закрепляя его при помощи балансировщика нагрузки). Ранее это приходилось делать самостоятельно, но теперь существуют некоторые инструменты, которые способны вам в этом помочь.
5. Тестирование приложения: здесь все аналогично. Благодаря самой сущности контейнеров весьма вероятно, что при такой модели ваши компоненты

будут небольшими; это значит, что у вас будет не очень много комбинаций конфигураций для осуществления тестирования. Адаптация вашего подхода в качестве к миру сменяющихся конфигураций будет крайне необходимым шагом, для того чтобы данная модель применялась успешно. Помните, что всякое тестирование связано с управлением рисками. Вам придется разработать подходящую стратегию, так как работа с релизами в традиционной манере (когда все изменения со временем все больше связываются друг с другом) не будет практичным подходом в рамках данной модели.

6. Визуализация процесса доставки: теперь вам необходимо визуализировать и оценить чуть больше аспектов, например количество контейнеров и их состояние, а вдобавок и создание новых контейнеров и возможности для развертывания, но в целом идеи остаются прежними.

Модель B – доставка с поддержкой контейнеров

При модели B приложения развертываются как ряд контейнеров на динамически создаваемых сетевых ресурсах

Основным отличием от модели Б выступает зрелость практик применения контейнеров и более модульная архитектура приложения

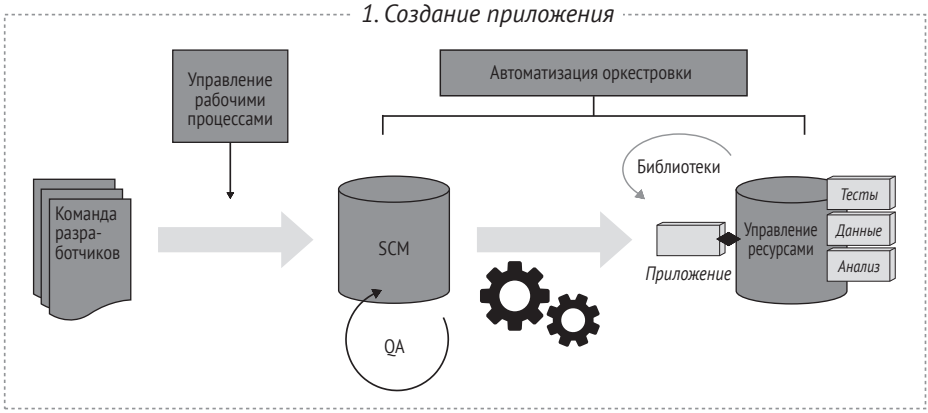


Рис. 9.3. Доставка с поддержкой контейнеров управляет приложением в контейнерах

Аспекты внедрения и воздействие на организацию

Сейчас вы имеете дело с неизменяемыми контейнерами, а стало быть, обслуживание контейнеров становится новой организационной обязанностью. Как вы проверите в случае обнаружения новых уязвимостей, в каком месте вы использовали определенные библиотеки, так чтобы можно было обновить все *образы контейнеров*? Так как вы выстраиваете контейнеры слоями, то, вероятно, используете старый образ с известными уязвимостями, расположившийся в некоторой цепочке или находящийся в публичном реестре. Вам придется обслуживать некоторые подходы в организации, чтобы вы могли обеспечивать использование ряда разновидностей контейнеров. Понятно, что, используя контейнеры, вы можете применять множество различных технологий одновременно, но организация должна обслуживать ряд допустимых подходов, так

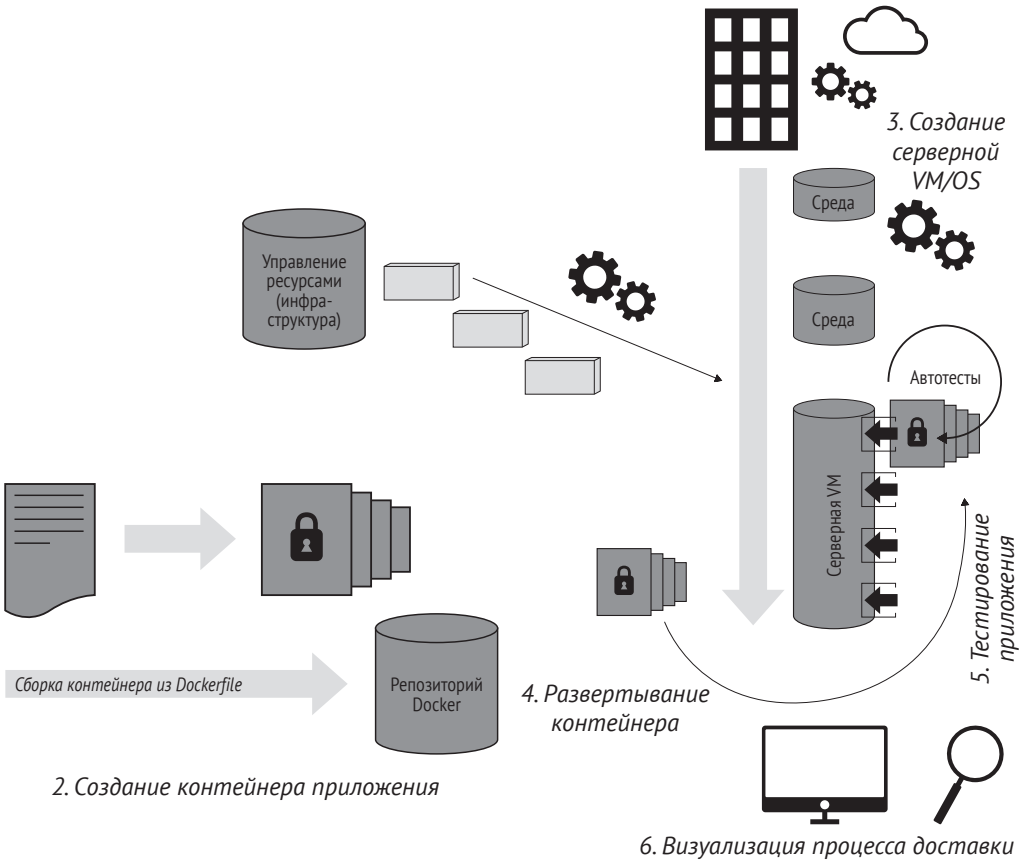


Рис. 9.3 (окончание)

как нужно поддерживать архитектуру, а предпочтения отдельных команд по технологиям могут впоследствии вызвать проблемы. Нахождение золотой середины – вот чем вы будете заниматься, практикуя данный подход в вашей организации.

Подобно облачным технологиям, работа с контейнерами подразумевает, что вам нужно перепроектировать архитектуру существующих приложений, чтобы использовать эту новую парадигму. Если просто поместить существующее приложение в большой контейнер, это не даст возможности в полной мере воспользоваться преимуществами доставки с поддержкой контейнеров. Вместе с этим проектированием необходимо также проводить некоторую реорганизацию, так как весьма неэффективно иметь контейнер приложения, которым занимается не одна команда. В лучшем случае один контейнер приложения должен обслуживаться одной командой. Если приложения действительно небольшие, то одна команда сможет обслуживать множество приложений. Но если контейнер приложения слишком большой для одной команды, то, возможно, он в принципе чересчур велик и его в дальнейшем стоит разбить на несколько частей. Заставьте закон Конвея работать на вас и создайте организационную структуру, которую вы хотели бы видеть в вашей архитектуре.

Оценка модели доставки: бессерверная доставка

Если вы незнакомы с тем, что такое бессерверная архитектура, поясню: это такая модель обслуживания, при которой вы не используете серверы как таковые, но просто пишете функцию, обеспечивающую для вас работу некоторой логики. Когда эта функция вызывается, создается экземпляр, и он существует, пока работает *вызов функции*. Примером такой архитектуры может послужить Amazon Lambda. Хотя некоторые организации, с которыми я работал, пробовали использовать эту архитектурную модель, я еще не наблюдал широкого ее применения. Вам, вероятно, было бы интересно изучить эту архитектуру и поискать модели применения, чтобы попробовать использовать ее в вашей организации.

Схема возможностей

Несмотря на то что всегда в схемах возможностей будут прослеживаться расхождения в силу контекстуальных различий, существуют общие приемы совершенствования ваших технических возможностей. Вам понадобится сначала иметь дело с системой управления конфигурациями (SCM) и сборкой приложений, чтобы сократить количество побочного шума, а затем нужно будет продолжить с развертыванием приложений. Если вы будете делать это в обратном порядке, вам придется много чего переделывать в автоматизации раз-

вертывания, так как *артефакты сборки* будут меняться, по мере того как вы будете осваивать автоматизацию процесса.

В идеальном случае стоит управляться с системой управления конфигурациями, сборкой приложений и автоматизацией развертывания одновременно. Для автоматизации тестирования потребуется наличие некоей предсказуемой среды, с которой можно было бы работать (например, в ней не должно быть проблем с конфигурацией или различиями приложений, обнаруживаемыми от развертывания к развертыванию), поэтому выгоду она принесет тогда, когда будут заранее автоматизированы сборка и развертывание приложений. Автоматизация подготовки среды обычно требует много времени на подготовку к реализации, поэтому вы можете начинать осуществлять ее параллельно, чтобы она была уже готова к тому времени, как понадобится. Другие же возможности добавляются после основных. Все это необходимо поддерживать при помощи постепенного выстраивания вашей DevOps-платформы, дабы иметь возможность способствовать вашей деятельности по автоматизации и эксплуатации, например мониторингу.

На рис. 9.4 изображена распространенная схема развития начального ряда возможностей. Заметьте, что необходимо провести некоторую конфигурацию инфраструктуры и организационное проектирование, перед тем как погружаться в выстраивание подходов к применению различных техник. Этот процесс последует сначала за системой управления конфигурациями и автоматизацией сборки, потом будет автоматизация развертывания, а затем последует шаблон автоматизации тестирования, что, согласно моему опыту, обеспечит самые высокие шансы на успех.

Первые шаги вашей организации

Приспосабливайте ваши модели доставки приложения

Как я уже говорил, нежелательно применять модель доставки с поддержкой контейнеров сразу ко всем вашим приложениям: это будет неэкономично и не вполне реализуемо. В организациях вы наверняка встретитесь с большим количеством legacy-приложений вместе с частью приложений, использующих непрерывную доставку и облачную доставку, а также частью приложений, использующих доставку с поддержкой контейнеров. И это реалистично. Помните, что нам нужно добиться улучшений, и слишком часто мы стремимся к идеалу, не обращая внимания на то, что принесет реальную пользу. Имейте это в виду и проведите практикум, на котором вы пересмотрите приложения и опишете то, что имеется на данный момент, а также идеальную модель доставки для каждого приложения. Вам понадобится для этого пригласить на общее совещание сотрудников, работающих с инфраструктурой, архитектурой и организацией доставки. Затем проанализировать возможности, необходимые для реализации моделей доставки, которые вы выбрали для каждого прило-

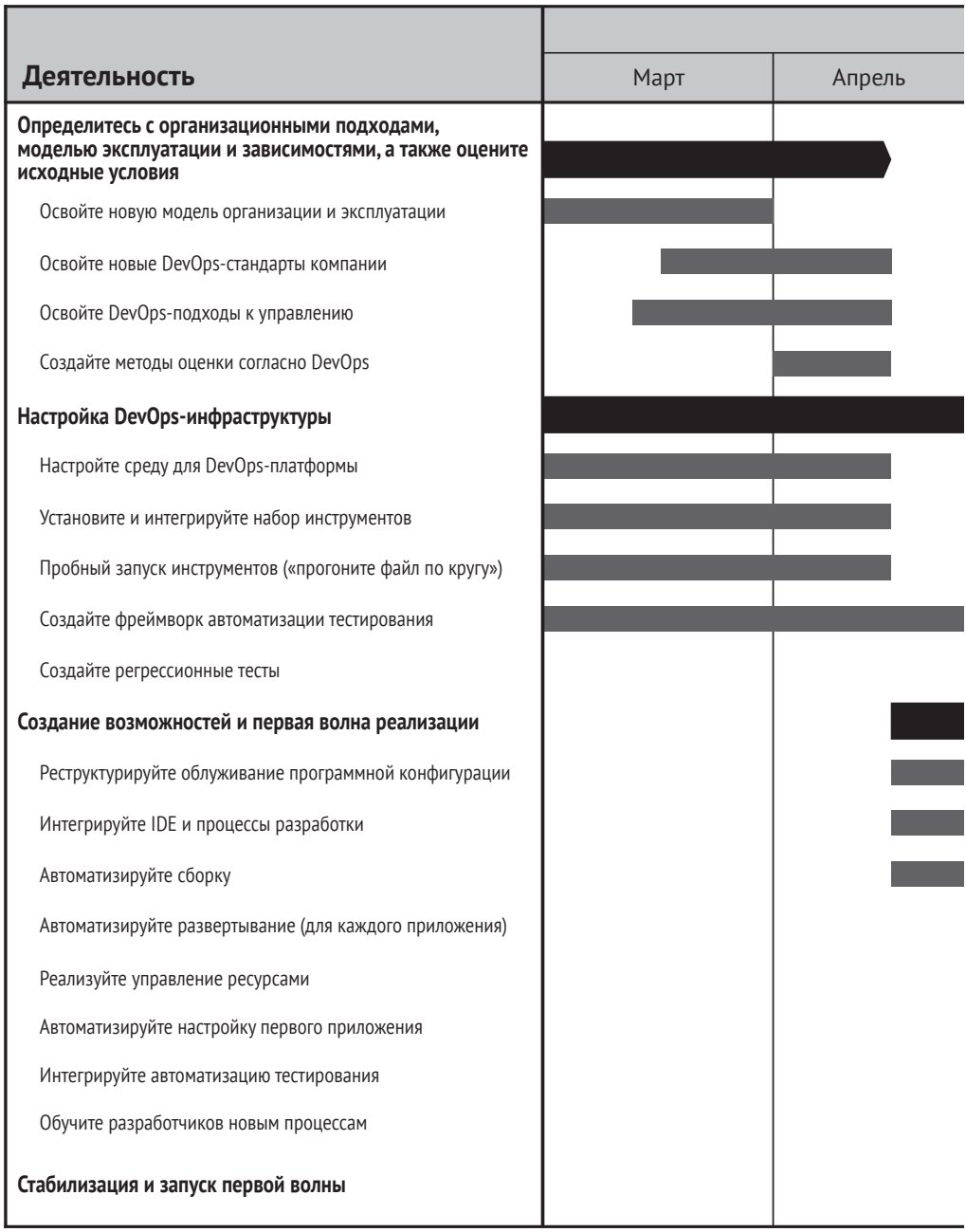


Рис. 9.4. Типовой план для начального набора возможностей: первыми шагами зачастую являются применение контейнеров и настройка инфраструктуры

Рис. 9.4 (окончание)

жения. Поразмыслите о ряде инициатив, которые помогли бы достичь недостающих возможностей. Вы сможете заново использовать возможности для приложений одного и того же технологического стека (например, Java-приложений), как только вы создадите их для первого приложения. Определитесь с возможностями, которые подходят для повторного использования. Учитывая все перечисленное, выстройте шестимесячную схему; пересматривайте схему и успех продвижения по ней ежемесячно, чтобы иметь возможность заново приоритизировать задачи, основываясь на полученном ранее опыте.

ГЛАВА 10

Архитектура приложений и микросервисы

Архитектура зависит от времени ее создания.

Людвиг Мис ван дер Рохе,
цитата из Программ и Манифестов
архитектуры XX века
авторства Ульриха Конрадса

Архитектура в последнее время играет все более и более важную роль. Когда я начинал работать в ИТ, архитектура приложения была чем-то статичным, описывала стек технологий и определяла организационную структуру (например, команду по работе с базами данных, Java-команду и интеграционную команду). Взаимодействие между получившимися командами выстраивалось согласно потоку данных в архитектуре, и архитекторы здесь были посредниками, регулирующими конфликты из-за расхождений во мнениях. Проекты архитектуры стремились к тому, чтобы достичь определенного конечного состояния, а выстраиваемые по ним диаграммы были сложными и довольно подробными, достигающими уровня модулей и вызовов функций. Подход довольно сильно изменился за последние несколько лет. Сегодня создание архитектуры основывается на принципах, и она преследует цель не достичь конечного состояния, а обеспечить гибкость, которая поможет приложению развиваться со временем. В идеальном случае она предоставляет командам возможность работать независимо друг от друга по мере внесения функциональных изменений.

В этой книге я много говорил о различных возможностях, которые вам нужно обеспечить в организации, а также о том, какие изменения ей нужны. И все же существует некоторая хитрость для создания архитектуры: архитектура приложения станет для вас основным препятствием, по мере того как вы бу-

дете ускорять темпы доставки. Если вы припомните какой-нибудь типичный проект, то, вероятно, заметите, что скорость осуществления доставки определяется самым медлительным компонентом. Это типичный признак сильно связанной архитектуры. В таких типах архитектур обычно предусматривается некоторое COTS-приложение, которое еще больше замедляет доставку из-за своей монолитности. Я соглашусь со словами Джеза Хамбла о том, что культура и архитектура являются двумя основными препятствиями для достижения высокой производительности [1].

В этой главе я хочу помочь вам рассмотреть свою архитектуру и найти способы ее развития. Согласно концепции *ИТ-изоморфизма бизнеса*, ваша архитектура должна отражать потребности бизнеса, что позволит командам непосредственно поддерживать бизнес, не прибегая к бесконечным переговорам, в которых принимают участие представители разных отделов организации. Очевидно, что это довольно тяжело осуществить и нам не нужно идти настолько далеко, но монолитные приложения нередко выступают наиболее подходящей моделью – опираясь на наши подходы, нам просто нужно найти золотую середину. Эта глава включает три раздела: в одном описываются архитектурные принципы, в другом – эволюция архитектур, а третий посвящен микросервисам (ни одна хорошая книга о DevOps сегодня не обходится без упоминания микросервисов, не так ли?).

Хорошая архитектура дается нелегко

С того времени как я начал работать в ИТ, я серьезно интересовался архитектурой. Мой отец был архитектором «в буквальном смысле», как он любит говорить (он проектирует здания), поэтому я могу продолжить его дело, хоть и в качестве ИТ-архитектора. Как и архитектура в классическом понимании, ИТ-архитектура не обходится без основополагающих принципов, необходимых для создания успешно работающей архитектуры, которая выдержит проверку временем. И подходы к построению архитектуры также развиваются – иногда благодаря моде, иногда из-за того, что мы узнаем нечто новое.

За первые два года моей карьеры в инструменты архитекторов было включено несколько общих принципов построения архитектуры:

- KISS – «Keep it simple, stupid» («Не усложняй, тупица», или, выражаясь деликатнее, «Будь краток и не усложняй»): предостережение от чрезмерного усложнения вашей архитектуры в расчете на ситуации, которые, с большой вероятностью, никогда не возникнут;
- DRY – «Don't repeat yourself» («Не повторяйся»): многоразовое использование поможет избежать синдрома «мое колесо круглее твоего» – вместо того чтобы 10 раз писать код, который делает одно и то же, нужно написать код, который можно будет использовать 10 раз и более;
- используйте слои: типичная трехуровневая архитектура была стандартом в течение долгого времени (уровень представления, уровень бизнеса, уровень данных);

- инкапсулируйте: исходя из объектно-ориентированных практик, позаботьтесь о том, чтобы клиенту сервиса не было необходимости узнавать, как работает сервис; обеспечьте слабое связывание архитектур.

На заре поры облачных вычислений архитектурные принципы развивались стремительно, пытаясь приспособиться к характеру облачных вычислений. Им приходилось меняться, чтобы извлекать максимальную выгоду из облачных вычислений, – чем и занималось большинство организаций. Для настоящих облачных вычислений используется следующее:

- самообслуживание по требованию. Нам нужно иметь возможность автоматически создавать инфраструктуру для наших сервисов, делая вызовы к API, чтобы предотвратить необходимость ручного вмешательства;
- объединение ресурсов. Причина тому, почему облачные вычисления могут обходиться дешевле, – другие еще используют то, чем мы уже не пользуемся. Выделенные ресурсы можно разместить в облаке, но это может ограничить возможную выгоду от стоимости размещения, если таковая вероятна;
- сетевой доступ. Нам нужно иметь возможность получать доступ к нашим сервисам из любой точки, а также оперировать сервисами с целью сокращения ненужных ресурсов и локализации трафика;
- быстрая растяжимость. Масштабирование с целью поддержки большего количества запросов к сервисам и сворачивание для сокращения стоимости использования должны осуществляться быстро;
- дозированное использование сервиса. Нам хочется платить только за то, что мы используем, иначе облако может обойтись довольно дорого.

К сожалению, существуют некоторые облачные сервисы, которые не поддерживают всех этих возможностей. Однажды я работал с облачным провайдером, который предоставлял в пользование новую машину, после того как вышлешь ему Excel-таблицу. Это кардинально расходится с первой идеей о самообслуживаемости; также это влияет на возможную стоимость, поскольку в таком случае между запросом на изменения и самими изменениями будет происходить задержка. Вы не можете обеспечить растяжимость, если полагаетесь на Excel-таблицы.

Что касается архитектуры, это те характеристики, к которым нам необходимо стремиться, чтобы автоматически предоставлять наши сервисы, – платить лишь за то, что нам нужно касательно инфраструктуры, и предоставлять качественные сервисы благодаря возможности использовать автоматическое масштабирование и сокращение ресурсов.

Ниже представлены некоторые архитектурные принципы, которым должны следовать приложения, работая с облаком:

- автоматизированное развертывание. Если при развертывании вам нужно прибегать к шагам, выполняемым вручную, то ваша автоматизация предоставления среды будет ограничена этими шагами, и, получается, вы просто переместите «бутылочное горлышко» на новое место;

- ограничение потребления ресурсов. Чтобы использовать сервисы дозированно, нам нужно отключить компоненты, которые нам не нужны, и всегда эксплуатировать их в нужном масштабе – не слишком большом и не слишком малом;
- независимость от расположения. С учетом того, что определенные экземпляры приложения могут перемещаться по сети, нашей архитектуре необходимо иметь возможность находить каждый из компонентов. Впоследствии это поможет нам выстраивать устойчивые архитектуры;
- независимость от инфраструктуры и поставщика облачного сервиса. Нам необходимо, чтобы мы не зависели от одного поставщика; это позволит нам следовать более разнообразным моделям сокращения ресурсов и производить поиски поставщика, который предлагает наилучшее соотношение «цена–качество». Заметьте, что в действительности смена провайдеров не всегда выглядит реалистичной; в таком случае вам необходимо максимально выгодно использовать экосистему выбранного вами поставщика;
- событийно-ориентированная архитектура. Взаимодействие должно происходить асинхронно, так как сеть нельзя считать надежным средством, это позволит приложению лучше справляться с сетевыми задержками. Синхронные вызовы потребляют ресурсы в режиме ожидания и могут стать причиной значительных проблем с производительностью;
- устойчивость к задержкам. Исходя из предположения, что мы не можем рассчитывать на гарантированный ответ на запрос, нужно учитывать, что нашим приложениям придется с этим сталкиваться и как-то справляться с воздействием задержек на пользователя;
- горизонтальная расширяемость. Нужно масштабировать архитектуру, предоставляя дополнительные модули. Значит, должна быть возможность распараллеливать наш процесс. Это позволит перенаправлять запросы к любому экземпляру, что, в свою очередь, увеличит устойчивость архитектуры;
- и конечно же, безопасность. Это вашим архитекторам надлежит продумать на всех уровнях архитектуры; нельзя перекладывать такую задачу на плечи поставщика облачного сервиса[1].

Совершенствование вашей архитектуры с течением времени

Большинство организаций на данный момент используют модель архитектуры, основанную на связанной сети систем, которые полагаются на готовые коммерческие продукты (COTS), универсальные технологии или нечто подоб-

[1] Еще один пример хороших принципов построения архитектур – двенадцатифакторные приложения. Их описание вы можете найти на сайте 12Factor.net (см. материал Адама Виггинса).

ное. Поверх этой сети систем располагается *уровень доступа*. Представление некоторой ценности этого уровня доступа для бизнеса сильно зависит от основных систем. В действительности же, перед тем как организация сможет с такой архитектурой внести значительные изменения, ей придется полностью перестроить монолитное приложение. Таково на данный момент положение дел во многих организациях.

Стратегия 1: декомпозиция

Одна из стратегий, используемых нами в работе с публичным сервисом клиента, который активно пользовался мейнфреймами, заключалась в том, чтобы произвести декомпозицию архитектуры и создать *уровень абстракции* между основной системой и уровнем доступа. В нашем случае мы дали задание нашей команде, работающей с мейнфреймами, создать *расходуемые сервисы*, которыми могла бы воспользоваться наша *фронтенд-команда*. Обе команды работали согласно Agile, и мы пользовались общими планировочными событиями (подобно PI-планированию из SAFe), для того чтобы согласовать, какие сервисы все же станут доступны в определенный момент. Фронтенд-команда могла независимо развиваться, в то же время используя сервис, предоставленный приложением мейнфрейма. Когда у нее появлялась необходимость в новом сервисе, эта задача приоритизировалась в бэклоге команды мейнфрейма. По сути, это была *двухскоростная модель доставки*, обеспечиваемая уровнем сервисов, который выступал в качестве связующего между двумя режимами работы команд.

Стратегия 2: переход на диету

Согласно этой стратегии, вы, по сути, обращаетесь к необходимости обеспечить гибкость и быстроту работы, сокращая свой технический долг и наделяя ядро вашей системы большей гибкостью. Стратегия считается весьма консервативной, но в определенном контексте, особенно там, где после массивной адаптации COTS-продуктов их стало трудно поддерживать, она способна предоставить вам пространство для маневра, когда вы будете принимать решения по принятию дальнейших шагов с применением другой стратегии при перестраивании вашей архитектуры.

В работе с одним из моих клиентов мы обратили внимание на состояние одного из Siebel-приложений, используемых нами, и на то, как оно развивалось с течением времени. Нам стало ясно, что мы накопили большой технический долг. Например, некоторые изменения, сделанные нами в этом приложении, уже были интегрированы в продукт в более эффективной реализации. Также множественные адаптации продукта привели к тому, что приложение стало тяжело поддерживать – в отдельных случаях силами одной команды было уже не обойтись. Мы решили избавиться от адаптаций и посадили приложение на диету. Со временем это позволило сократить длительность цикла доставки, стоимость внесения изменений и помогло сделать код более поддерживаемым.

Стратегия 3: давайте заодно это попробуем

Я заметил рост популярности этой стратегии в течение последних нескольких лет. Вместо того чтобы работать над основными системами, создают альтернативную архитектуру, предназначенную для обслуживания определенных бизнес-целей. Эта новая архитектура выстраивается «с нуля» (с самого начала и сразу без legacy-приложений) и понемногу интегрируется со старыми основными системами или старым уровнем доступа. В этой новой системе используются современные архитектурные принципы и технические практики. Зачастую для ее поддержки создаются отдельные организационные структуры (например, цифровые команды).

Один из моих клиентов создал цифровую команду, которая работала принципиально иначе, нежели прежние технологические команды, поддерживавшие работу основных систем. В определенный момент испытанием стала необходимость интегрировать две части бизнес-логики и убедиться в том, что хорошие практики их новой организационной структуры и архитектура были переняты также и для старых основных систем, или же нужно было постепенно добавлять все больше и больше функциональности в новую организационную структуру, согласно паттерну заслонки, пока не появится возможность отказаться от старых систем и организационных структур. В случае с нашим клиентом мы пошли по пути интеграции и использовали проверенные способы работы с применением новой архитектуры, чтобы поддержать работу существующих legacy-приложений и ускорить доставку.

Стратегия 4: разрушайте основные системы при помощи микросервисов

Это самая популярная стратегия на данный момент. Суть в том, что основные системы выводятся из строя создаваемыми микросервисами, которые поддерживают части бизнес-логики таким образом, чтобы все меньше и меньше функциональности обслуживалось основными системами. Мы, опять же, используем шаблон заслонки, чтобы медленно продвигаться к достижению микросервисной архитектуры. Следующий раздел этой главы я посвящу микросервисам, чтобы помочь вам понять, как их стоит использовать.

Знакомство с микросервисами

Как я уже упоминал, микросервисы стали невероятно популярными, хотя само понятие микросервисов часто используется неоднозначно, так как не существует хорошего и точного определения, полноценно характеризующего микросервис. В целом вы можете считать, что микросервисы – это полная противоположность монолитных приложений.

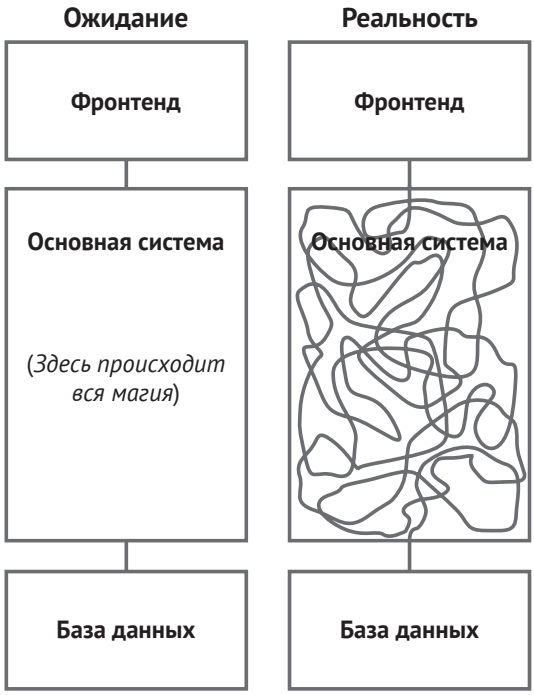


Рис. 10.1. Ожидание и реальность:
простые диаграммы не гарантируют простой архитектуры

Давайте немного поговорим о монолитных приложениях, ведь в некоторых случаях стоит использовать именно их. Если у вас сравнительно простое приложение, применение микросервисов для него будет лишней нагрузкой. В монолитном приложении зачастую легче исправлять дефекты и проводить тестирование, так как у него меньше подвижных частей, которые могут создать проблемы. И давайте признаемся, монолиты намного легче описывать и обслуживать – развивающаяся микросервисная архитектура с сотнями микросервисов, независимо разрабатываемых и развертываемых, принесет многим организациям проблемы с подходами к управлению ими. Если у какого-то компонента не будет необходимости в развитии, вы не сможете оправдывать разбитие его на микросервисы. Стоимость проектирования предсказуемой архитектуры микросервисов больше, чем у монолитного приложения, из-за внедрения дополнительных непостоянных обстоятельств, таких как сетевые задержки, совместимость с предыдущими версиями и поддержание очередности передаваемых сообщений.

С другой стороны, в то время как монолитные приложения снаружи выглядят прекрасно и отлично ведут себя на диаграммах, они часто представляют собой облачко с надписью «здесь происходит магия». Это означает, что вся сложность скрывается в «черном ящике». Я часто встречался с кодом Siebel

и немного SAP, который убедил меня в том, что за кажущейся простотой таятся большие сложности. И конечно же, все становится еще сложнее с ростом монолита и вовлечением все большего количества людей в его развитие.

Микросервисы помогают видеть сложность. Как говорил, например, Рэнди Шупа: «Микросервисы есть не что иное, как хорошо сделанное сервис-ориентированное приложение» [2]. По факту, это и есть несколько сокращенное определение хорошего микросервиса: это сервис (приложение), который служит одной цели, он изолированный и независимый, у него четко определенный интерфейс и изолированная среда обслуживания (вплоть до наличия отдельной базы данных для каждого из сервисов).

Ниже приведена полезная аналогия, которую я услышал впервые из речи Джеймса Льюиса:

> *Это, милорд, фамильный боевой топор. Мы владеем им уже почти девять столетий. Конечно же, иногда приходилось менять лезвие. Иногда нужна была новая рукоять, по-новому спроектированные металлические детали и свежие орнаменты... Но ведь это все тот же фамильный девятисотлетний топор. Именно постольку, поскольку он мало изменился за все эти девять столетий, он все еще остался довольно неплохим топором, знаете ли. Довольно неплохим* [3].

Смена деталей топора поспособствовала тому, что по прошествии девяти столетий он все еще остался хорошим топором, и того же вы хотите в отношении архитектуры. Вместо того чтобы каждые несколько лет вести проекты по ИТ-трансформации, которые несут с собой большую долю риска и дезорганизации, вам нужно добиться постоянного развития частей вашей архитектуры, чтобы она естественным образом развивалась сама.

Каковы же преимущества применения микросервисов?

Со временем все в сфере ИТ начали понимать, что не существует конечной архитектуры. Архитектура ваших систем постоянно развивается, и к тому времени, как заканчивается реализация одной архитектуры, уже появляются идеи по поводу новой. Ранее обновление архитектуры до новой версии было нелегким достижением, так как приходилось заменять крупные системы. С приходом микросервисов начали создаваться экосистемы архитектур, которые позволяют вам все время менять малые компоненты и избегать масштабных переходов. Такая гибкость может заметно повысить скорость развития архитектуры. В дополнение структура микросервисов позволяет командам получить больше контроля над их сервисом, и это поможет вашим командам быть более продуктивными и ответственными в процессе разработки сервисов. Развертывание архитектуры и механизм релиза можно будет поддерживать намного легче, вам не придется беспокоиться о зависимостях, которые должны присутствовать при релизе и развертывании сервисов. За этим, конечно, последует усложнение процесса тестирования, так как у вас будет множество

вариантов взаимодействия с сервисом. Здесь важно использовать автоматизацию и разумные подходы к стратегии тестирования.

Почему стоит использовать микросервисы?

Микросервисы стоит применять в областях, в которые компания собирается вкладываться в будущем. Сферы, где важна скорость выхода продукта на рынок, хорошо подходят для начала использования микросервисов, потому как быстрота работы с ними – основное их преимущество. Архитектура, выстроенная на зависимостях, часто застывает в бездействии по ряду причин, начиная с того, что разработчикам нужно время на ознакомление со всеми зависимостями их кода, и заканчивая тем, что релиз компонентов будет откладываться из-за увеличения длительности его цикла. Микросервисам не сопутствуют эти проблемы благодаря их независимости, которая обеспечивается их архитектурой. Конечно, это утверждение верно до тех пор, пока вы пользуетесь хорошими практиками проектирования архитектуры для ваших микросервисов; никто не помешает вам создавать небольшие интегрированные сервисы, которые мы не сможем назвать микросервисами.

Еще одна область, на которую стоит обратить внимание, – приложения, у которых нет возможности вертикального масштабирования по экономическим соображениям. Возможности микросервисов для *горизонтального масштабирования* позволяют найти более экономичное решение для обеспечения расширяемости. Разумеется, стремление к использованию микросервисов потребует вложений, поэтому вам стоит искать области, в которые есть смысл вкладываться и где задачи, упомянутые ранее, обеспечат хороший старт для ваших начинаний.

Чего может стоить успешное применение микросервисов?

Это, конечно, неудивительно, но я все-таки замечу: множество вариаций дополнительной сложности, которая сопутствует независимо развертываемым сервисам, могут присутствовать также и в среде эксплуатации. Следовательно, вам действительно стоит знать, что вы делаете. Здесь я имею в виду то, что вам необходимо иметь зрелые взгляды в ваших практиках проектирования и достаточно хорошо налаженный конвейер развертывания, в котором автоматизировано «все» (непрерывная интеграция, развертывание, тестирование). Иначе усилия и сложность, создаваемые для целей поддержки всей этой конструкции вручную, вскоре перевесят выгоду использования микросервисов. Я работал с клиентом, у которого был небольшой ряд микросервисов (менее десяти), но возможности проектирования его не были настолько зрелыми для того, чтобы обеспечить дальнейшее масштабирование, так как все еще приходилось вручную собирать и тестировать эти микросервисы. Моему клиенту было сложно добиться выгоды от применения микросервисов, пока его команда не смогла улучшить свои практики проектирования.

Закон Конвея гласит: системы подобны той организационной структуре, в которой они создаются [4]. Следовательно, чтобы пользоваться микросервисами, нам необходимо освоить принципы Agile и DevOps о кросс-функциональных командах (и в идеальном случае еще и развивать их соответственно вашим потокам ценностей). Такие команды смогут иметь полноценное управление микросервисами, создаваемыми ими (от начала до конца). Подобный подход имеет смысл, если сервисы сами по себе небольшие и изолированные: если в работу с ними будет вовлекаться множество команд (DBA, .NET-разработчики и т. п.), вы только перегрузите работу с этими маленькими сервисами. Итак, микросервисы представляются мне следующим шагом к совершенствованию возможностей проектирования при помощи практик DevOps и Agile, так как они подразумевают, что организации уже освоили обе методологии (или хотя бы близки к этому).

Как можно начать применять микросервисы?

Если ваша организация уже готова к тому, чтобы начинать применять микросервисы (предпосылками к чему могут служить освоенные практики DevOps и Agile), то вперед: выбирайте пилотный проект и создайте микросервис, отвечающий рассмотренным выше характеристикам (один сервис – одна задача, изолированность, независимость, которая четко задается интерфейсом и изолированным обслуживанием) и несущий реальную ценность для бизнеса (например, нечто часто используемое, с чем нередко сталкивается клиент и что соответствует нашему стеку технологий). Ваш первый пилотный проект не принесет сиюминутного успеха, но этот опыт вам пригодится. Для работы с микросервисами потребуются вложения организации, и изначальная стоимость затеи может оказаться не столь очевидной (например, с самого начала может показаться, что добавление новой функциональности в монолитное приложение не будет требовать много усилий и средств), но в долгосрочной перспективе гибкость, быстрота и устойчивость микросервисной архитектуры изменят ваш ИТ-ландшафт. Окажетесь ли вы в конечном счете обладателем идеальной микросервисной архитектуры? Наверное, нет. Но ваши основные сервисы могли бы просто мигрировать в архитектуру, которая создается и проектируется таким образом, чтобы обеспечивать расширяемость, что пригодится вам в мире постоянно меняющегося рынка.

Первые шаги вашей организации

Определите свою стратегию развития архитектуры

Пригласите архитекторов на совещание о стратегиях создания архитектур. Позвольте им описать существующий план, а также попытайтесь наложить эти представления на стратегии развития архитектур, которые

упоминаются в данной главе: декомпозиция имеющейся архитектуры, сокращение технического долга, создание новой архитектуры на стороне или избавление от старой основы архитектуры. Затем обсудите альтернативные подходы и посмотрите, получится ли прийти к согласованной стратегии, которая со временем обеспечит еще большую декомпозицию сервисов. Убедитесь, учитывая стратегию развития архитектур, что обеспечивается еще и наличие возможностей, необходимых для ее осуществления. Пользуйтесь моделями доставки и необходимыми для них возможностями, чтобы помочь вашим архитекторам говорить не просто о каких-то набросках системы, но о действительной возможности поддержки архитектуры благодаря применению оптимальных методов проектирования.

ГЛАВА 11

Эффективное управление приложениями и применение DevOps-инструментов

> Не все, что нужно учитывать, можно сосчитать, равно как не все, что можно сосчитать, нужно учитывать.
>
> *Уильям Брюс Кэмерон, «Неформальная социология: Легкое введение в социологическое мышление»*

В предыдущих главах я уже рассказывал про модели доставки и про то, как (по моему опыту) люди тратят уйму времени на аспекты разработки. Притом основное внимание уделялось «Dev»-аспектам в рамках DevOps. В этой главе мы поговорим об эксплуатационных аспектах. Здесь будут затронуты три темы: как выглядит современная эксплуатация приложений (включая мониторинг и поддержку приложения), что подразумевает процесс обслуживания платформы и как DevOps-возможности способствуют уменьшению объема работ для каждой осуществляемой доставки, который экономически целесообразен и сокращает риски, связанные с эксплуатацией.

Современная эксплуатация приложений

В организациях принято считать поддержку приложения в продуктивной среде вынужденным злом, и ее осуществляли с применением устаревшего подхода, заключавшегося в том, чтобы найти поставщика, который согласился бы делать это по низкой цене. В конце концов, вам нужно просто обслуживать приложение – изменений, должно быть, не будет слишком много. Это, к сожалению, плохой подход к поддержке приложений: в таком случае они со временем будут изнашиваться. Вам нужно будет прикладывать больше усилий для их поддержки, и это только поспособствует накоплению технического долга. Однажды я услышал о правиле 50 %, и эта идея мне понравилась: если вы тратите более 50 % усилий команды по эксплуатации на «тушение пожаров» в продуктивной среде, то вы уже проиграли. Теперь вы будете тратить на это все больше и больше времени, а все из-за того, что поленились своевременно отладить автоматизацию и поработать с базой кода приложения для обеспечения поддержки.

Мне кажется, что с развитием веб-сервисов немного сменился центр тяжести, и большая доля поддержки стала важным аспектом для бизнеса. В итоге со временем ведение эксплуатации стали возвращать обратно в руки штатных команд, хотя многие организации, работающие по старинке, все еще усиленно вкладываются в новые возможности без улучшения процессов эксплуатации.

Приложения, работающие в продуктивной среде, ранее обслуживались весьма оперативно: когда появлялась проблема, кому-то на пейджер прилетало сообщение, и проблему начинали решать. В промежутках между такими вызовами велась работа над списком известных проблем, или вообще время, затраченное на решение проблем, восполнялось за счет нерабочего времени. Сейчас в организациях развивается новая культура, заметно отличающаяся от прежней: она рассматривает дефекты в продуктивной среде не как проблемы, а как возможность все больше улучшать работу с продуктивной средой. Продемонстрирую это на примере.

Как-то мы с одним моим другом из DevOps-сообщества сидели в кофейне в Портленде, заказали себе кофе. Пока наш заказ готовился, мой друг получил сообщение о том, что возникла проблема в продуктивной среде. Я ожидал, что он сейчас вскочит и начнет всем названивать, как принято во многих организациях. Но ничего подобного! Когда я осторожно поинтересовался, как дела, друг дал понять, что для начала мы попьем кофе. И не успели мы опорожнить свои чашки, как он получил сообщение о том, что все в норме! Я был впечатлен и сказал ему, что у него, вероятно, прекрасная команда, которая так быстро решает проблемы. Его ответ меня удивил. Напротив, сказал друг, команда довольно медлительная. И медлит она не случайно, а с целью узнать, что же произошло не так. На основе полученной информации сотрудники определяют, какие способы автоматизированного распознавания проблемы можно применить и какие автоматизированные способы смогут решить проблему; только после этого реализуется само решение. В этом случае проблема являлась некой

сущностью, которая автоматически исправлялась системой. Затем мой друг объяснил, что у организаций есть два выбора – «праведный» путь, на котором все больше времени можно использовать для активного совершенствования системы, и порочный путь, на котором вы решаете возникающие проблемы, не предпринимая попыток улучшить систему в целом. Я знаю, о чем говорю, – с тех пор я стал использовать этот подход в своих проектах.

В развитом состоянии системы среды эксплуатации должны обслуживаться таким образом, чтобы вмешательство человека требовалось по минимуму.

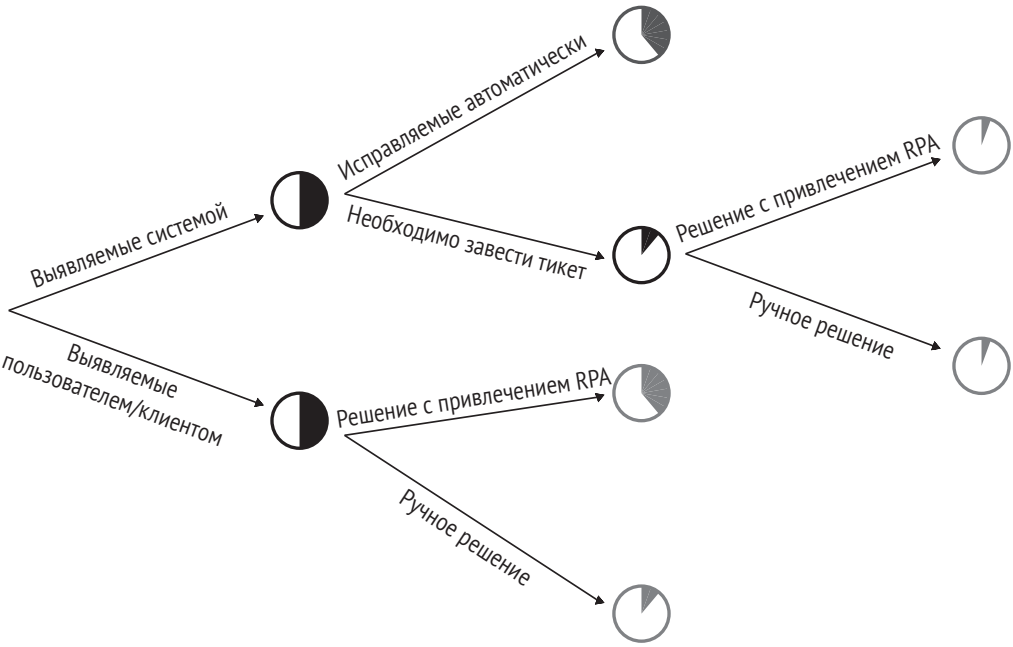


Рис. 11.1. Развитое состояние: современные системы работают по принципу сокращения объемов необходимой работы

Рассмотрим рис. 11.1. При помощи правильных подходов к мониторингу нам нужно достичь возможности самим выявлять только половину проблем посредством работы системы, чтобы эти проблемы не выявлялись в первую очередь пользователями («Выявляемые пользователем/клиентом»). Для большинства из них даже не понадобится заводить тикет – проблема будет переадресована получателям, созданным командой ранее, и разрешена автоматически («Исправляемые автоматически»). Для других проблем, которым мы еще не нашли автоматизированного решения, нужно заводить тикет, который, опять же, можно создавать автоматически. Многие сегодня считают, что мы можем обеспечивать решение этих проблем при помощи таких вещей, как роботизированная автоматизация процессов (решение с привлечением RPA),

чтобы свести к минимуму риск переделывания работы, а также объемы работы, выполняемой вручную (много сил зачастую уходит на выполнение небольшого ряда типовых задач, решение которых можно автоматизировать при помощи RPA).

Проблем, для которых не получится изобрести автоматизированное решение, останется не очень много, и они будут переданы команде специалистов по эксплуатации и будут разрешаться вручную. Что касается проблем, выявляемых пользователями, здесь мы прибегаем к схожей логике: например, пытаемся как можно чаще поддерживать разрешение проблем при помощи RPA и минимизировать количество проблем, для решения которых требуется много ручной работы. Есть еще пара полезных инструментов, которые стоит рассмотреть и использовать. Первый – это функциональность самообслуживания, которая позволяет пользователям осуществлять первую ступень поддержки и, возможно, автоматически запускать процессы напрямую из ресурса для самообслуживания. Второй – совершенствование при помощи искусственного интеллекта. При возникновении и решении проблем создается множество данных, и использование этих данных для глубокого обучения поможет вашим сотрудникам находить первопричины проблем и способы их разрешения. Все это еще находится на начальных этапах развития, но, поглядев, как компании начинают это реализовывать, я ожидаю значительного прогресса в ближайшие годы.

Чтобы воспользоваться подобными преимуществами, вам нужна достойная архитектура для мониторинга: необходимо наблюдать за инфраструктурой, производительностью приложения, а также его функциональностью. Но ведь больший объем мониторинга повлечет за собой и большую ответственность. Таким объемом информации людей можно попросту перегрузить. Обширный мониторинг – это хорошо, так как при этом предоставляются данные, а они лежат в основе всякого анализа и будущей автоматизации. В то же время огромный поток уведомлений осложняет работу. Поэтому настраивайте уведомления гибко, – не будите сотрудников отдела эксплуатации посреди ночи, если недоступность сервера на самом деле не повлияет на взаимодействие пользователя с ресурсом, или если у проблемы найдутся автоматизированные решения[1].

Формирование DevOps-платформы и работа с ней

В работе с DevOps-платформой также стоит учитывать деятельность, связанную с эксплуатацией, с которой нужно обходиться с неменьшей серьезностью. Мне нравится, как Open Group назвала одну из своих моделей IT4IT (ИТ

[1] Книга Бетси Бейер, Криса Джонса, Дженнифер Петофф и Найала Ричарда Мерфи *Site Reliability Engineering: How Google Runs Production Systems* («Проектирование надежности сайта: Как Google запускает производственные системы») описывает хорошие практики мониторинга, которые помогут вам сделать правильный выбор.

для ИТ) [1], и здесь это название как раз очень подходит. Вы работаете с ИТ-системами, которые поддерживают ИТ-системы бизнеса. Поддержание их в рабочем состоянии не менее важно, чем сама работа систем продуктивной среды, и это не должно осуществляться просто с рабочей машины разработчика или в «песочнице». Представьте себе ситуацию, когда вы сталкиваетесь с инцидентом в продуктивной среде, который требует срочных исправлений. Если в это время будет недоступна или повреждена система управления версиями исходного кода, то у вас будут неприятности! Проектировать DevOps-платформу нужно с учетом этих эксплуатационных сложностей.

Подобно обсуждаемому выше выбору приложений для бизнеса, в выборе инструментов для DevOps у нас такая же проблема[1]. Какие инструменты лучше всего подойдут для DevOps? Не буду углубляться в описание конкретных инструментов; вместо этого я расскажу вам, на что обращаю внимание при поисках DevOps-инструментов, кроме предоставляемой ими функциональности. По моему опыту, составлять хорошие наборы DevOps-инструментов можно из любых инструментов, но некоторые инструменты сложнее интегрировать, чем другие.

Очевидно, что DevOps-инструменты должны соответствовать DevOps-практикам и способствовать формированию правильной культуры. Это означает, что инструменты должны работать за пределами своей экосистемы. Очень маловероятно, что компания будет пользоваться инструментами только одного поставщика или экосистемы. Поэтому наиболее важное качество инструмента – возможность интеграции с другими инструментами (а также возможность последующей его замены, что очень важно для выживания на стремительно развивающемся рынке). Получается, в первую очередь при поисках DevOps-инструментов стоит обращать внимание на то, как хорошо поддерживается API. Можете ли вы пользоваться всей функциональностью, предоставленной UI, посредством API (при помощи консоли или языка программирования)?

Мы должны обращаться с нашими инструментами, как и с любым из наших приложений в организации, то есть нам нужно применять к ним контроль версий. Второе, на что стоит обратить внимание, – возможность управлять версиями всех конфигураций инструмента во внешнем конфигурационном файле (не только в самом приложении). Со вторым аспектом связана функциональность для поддержки множества окружений для инструмента (например, разработки и эксплуатации). Насколько легко будет передавать конфигурации по средам? Как можно проводить слияние конфигураций в различных окружениях (или строках кода)? Необходимо, чтобы у всех в компании был один и тот же инструмент. Это уже связано с поисками подходящей модели приобретения лицензии. Конечно же, есть и открытое программное обеспе-

[1] Кажется, что новые DevOps-инструменты появляются на рынке чуть ли не каждый месяц. Это можно объяснить тем, что бывает трудно описать все используемые DevOps-инструменты. Советую обратить внимание на XebiaLabs – один из лучших справочников, своеобразную «периодическую таблицу» DevOps.

чение, которое прекрасно справляется с такой задачей, но как насчет коммерческих решений? Они тоже могут пригодиться! Еще важнее, что они не отталкивают от их использования. Например, для инструментов, которым необходимо пользоваться агентом, не должна выставляться оплата за каждого агента, так как в таком случае будет присутствовать желание не использовать их повсюду. Проводите переговоры о корпоративном решении, чтобы для каждого случая использования не приходилось создавать бизнес-кейс.

Визуализация и аналитика – важные аспекты каждого набора DevOps-инструментов. Чтобы они работали, нам нужно добиться легкого доступа к данным; это значит, что у нас будет необходимость извлекать данные или запрашивать их. Если ваши данные хранятся в непригодной модели данных или если у вас нет способа получения доступа к данным и извлечения их для анализа и визуализации, то потребуются дополнительные время и усилия для получения подходящих данных. Панели управления и отчеты, формируемые самим приложением, не могут послужить достойной заменой, ведь вам наверняка понадобится проводить агрегации и анализ в рамках набора инструментов.

Полагаю, эти критерии уже достаточно очевидны, но, как ни удивительно, немногие инструменты им соответствуют. Инструменты open source зачастую выглядят выгоднее в этом плане, но для работы с ними наверняка понадобятся хорошие технические навыки членов вашей команды, чтобы она могла осуществлять настройку и поддержку. Я надеюсь, что поставщики инструментов начнут понимать: если они хотят предоставлять DevOps-инструменты, нужно прислушиваться к ценностям, принятым в культуре DevOps, иначе они так и будут проигрывать инструментам open source. В то же время подумайте о том, что более всего важно для вас и вашей организации, а затем примите компромиссное решение, поступившись малозначимым критерием. Не существует идеальной архитектуры инструментов.

Но как вы будете управлять постоянно развивающимся пространством DevOps-инструментов? Здесь я бы посоветовал применять практический подход, так как вам нужно будет все стандартизировать и в то же время сохранить гибкость. В общем случае в крупной организации имеет смысл иметь минимальный набор инструментов для поддержки, и тому есть несколько предпосылок:

- оптимизация стоимости приобретения лицензии;
- навыки сотрудников, которые можно будет использовать по всей организации;
- упрощение интеграции инструментов.

Да, с одной стороны, некоторые инструменты лучше других подходят для специфичного контекста (например, .NET-инструменты могут отличаться от ваших основных инструментов). К тому же постоянно появляются новые инструменты.

В целях сохранения гибкости я реализовал следующий подход:

- начинайте с небольшого набора стандартных инструментов в вашей организации;
- позвольте определенной доле команд постепенно отказываться от использования стандартных инструментов (к примеру, в течение трех–шести месяцев);
- в конце этого «пробного периода» соберите информацию и вынесите решение о том, что делать с данным инструментом:
  - заменить его стандартным;
  - добавить его к ряду инструментов для применения в специфичном контексте;
  - отказаться от использования нового инструмента, а затем перевести команду обратно на стандартный инструмент.

Все, что мы ранее говорили про поддержание работоспособности систем продуктивной среды, применимо и к нашей DevOps-платформе. Об этом я уже вел речь в разделе о моделях доставки; вам нужно определиться с топологией окружений, которая позволит развивать вашу DevOps-платформу вместе с окружениями для разработки и эксплуатации. Как видите, понятие IT4IT от Open Group довольно точно описывает суть дела.

Работа с небольшими партиями изменений

Один из наиболее важных факторов, которые влияют на прикладываемые усилия и риски в среде эксплуатации, – размер поставляемой партии изменений. Ранее партии были довольно большими, так как множество изменений было сведено в ежегодном или ежеквартальном релизе. Так обстояло дело, поскольку верилось в то, что менее рискованно иметь дело с редкими, но крупными изменениями, к тому же действовали экономические мотивы: при таких условиях стоимость этого события была меньше. Оптимальный размер партии определялся по стоимости развертывания партии и по предполагаемой выгоде от удержания завершенной функциональности до наступления времени следующего релиза (см. рис. 11.2). Поскольку стоимость развертывания была высока (в расчет принимались контроль качества, исправление проблем и продуктивная среда после развертывания), объем партии приходилось делать большим, но методы, представленные в третьей части книги, помогут вам сократить операционные расходы и, таким образом, уменьшить стоимость партии изменений (см. рис. 11.2). Это, в свою очередь, увеличит скорость осуществления доставки, упростит внесение каждого изменения, а также уменьшит общую стоимость развертывания и работы с изменениями в среде эксплуатации. Поэтому кроме выгоды для бизнеса, о которой мы говорили выше, малые партии обеспечивают еще и практические преимущества в плане осуществления эксплуатации.

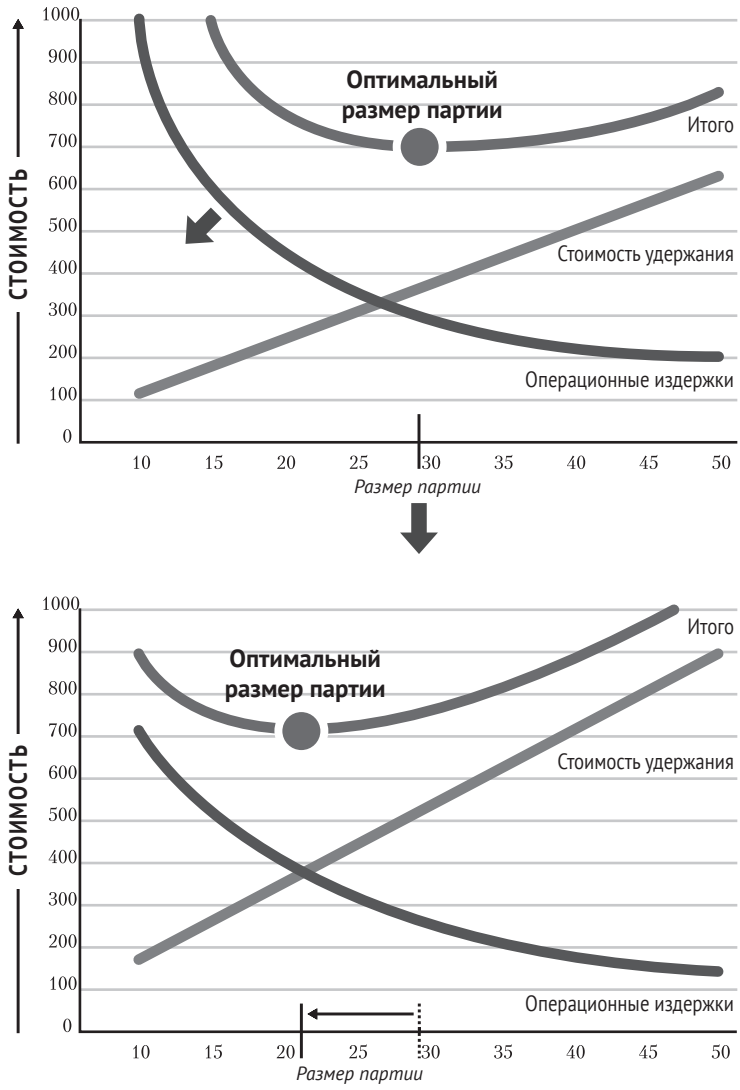


Рис. 11.2. Сокращение операционных издержек способствует уменьшению размеров партий с изменениями

Первые шаги вашей организации

Анализ проблем

В этом упражнении мы рассмотрим способы улучшения администрирования вашего приложения при помощи анализа тикетов с проблемами, чтобы определить, что можно автоматизировать. Как я уже говорил, у вас

есть уйма данных, потенциал которых не используется. Примитесь за эти данные из тикетов о проблемах и проведите несложный их анализ. Удобно будет использовать облако тегов, чтобы разметить общие понятия (например, «перезапуск сервера», «смена пароля», «нет данных»), а затем провести категоризацию на основе этих данных. Разобравшись с этим, пробегитесь по тем категориям, которые упоминались чаще всего, чтобы увидеть, что можно улучшить в системе автоматически или хотя бы с частичным использованием автоматизации. Обычно требуется два-три отборочных цикла перед осуществлением категоризации, основанной на наборе тегов и других метаданных, – этого будет достаточно, чтобы составить относительно точную картину для анализа. Это даст вам начальную точку для создания автоматизированной системы среды эксплуатации, которая со временем сможет сама себя исправлять.

Пересмотрите свои DevOps-инструменты

Подкрепитесь хорошими принципами выбора DevOps-инструментов (достойный API, конфигурация как код, подходящая модель получения лицензии), посовещайтесь с архитекторами в вашей организации и составьте список инструментов, которые вы используете для поддержания ваших DevOps-возможностей. Вы удивитесь, как много инструментов у вас есть и сколь многие из них частично перекрываются по функциональности! Проанализируйте эти инструменты на предмет того, как долго они смогут служить (при помощи табл. 11.1, куда вы можете добавить свои критерии, специфичные для вашего контекста), а также выявите слабые места, где ваши инструменты действительно не соответствуют методологии DevOps и тянут вас назад. Сформируйте стратегию замены этих инструментов в будущем.

Таблица 11.1. Обзор DevOps-инструментов: они должны сами соответствовать хорошим практикам DevOps

Критерий	Инструмент А	Инструмент Б
Поддержка API		
Управление конфигурациями		
Множественность окружений / поддержка ветвлений кода		
Модель приобретения лицензии		
Доступ к данным		

ГЛАВА 12

Облако

Джей: Оно отправилось! Оно отправилось в облако!
Энни: И ты не можешь спустить его обратно из облака?
Джей: Никто не понимает, как работает облако! Это чертова загадка!

Из фильма «Домашнее видео: только для взрослых»

К сожалению, для многих организаций облако все еще является чем-то загадочным – точнее говоря, не сама концепция облака, а выгода, которую облако может предоставить. В данной главе я хочу приоткрыть завесу тайны и рассказать про некоторые сложности, которые могут возникнуть при переходе к работе с облаком, а также про то, что нужно сделать организации, чтобы в дальнейшем пользоваться преимуществами облака. Итак, ниже мы сосредоточимся на переходе на облачную инфраструктуру и управлении ей. Мы уже обсуждали понятие «программа как сервис» (SaaS) в главе 3, а также говорили об архитектуре приложения в главе 10. Все это поможет вам осмыслить многоплановое применение облачных сервисов.

Данная глава, по большому счету, раскрывает выгоду их использования, но не стоит забывать и о других преимуществах и рисках, связанных с облаком. Из плюсов: легче настраивать сокращение ресурсов, что поспособствует их устойчивости масштабируемости, а также прочной экосистеме связанных между собой сервисов, которую вам так или иначе пришлось бы выстраивать самостоятельно. Из минусов: зависимость от стороннего поставщика, сложности с принадлежностью данных, а также риск атак в том случае, если вы пользуетесь популярной платформой.

Базовые принципы облачной экономики

Прежде чем мы начнем обсуждать переход на облако, возможно, стоит подробнее ознакомиться с экономической моделью, по которой работает облако. Если вы обратитесь напрямую к сравнению облачных сервисов и локальных решений, то в общем случае облачные решения окажутся более дорогостоящими (например, наличие одного полностью используемого локального сервера обойдется дешевле, чем наличие такого же активно используемого сервиса в облаке). Некоторые даже моделировали соответствующие ситуации, и вы можете поискать исследования подобного рода в интернете. Хотя осуществлять подобное сравнение в принципе проблематично, так как стоимость поддержания экосистемы высчитать сложно. По опыту работы с клиентами могу сказать, что перенос приложений в облако часто не приносит значительного сокращения расходов, пока вы не посвятите время некоторому рефакторингу и перепроектированию архитектуры.

Экономика облака основана на общем использовании ресурсов, чтобы все могли платить только за то, чем они пользуются, а остальные ресурсы могли использоваться кем-то другим. С другой стороны, это означает, что часто меняющиеся сервисы смогут в большей мере воспользоваться преимуществами облачной модели. Преимущества облачной модели зависят от того, насколько быстро сервис способен адаптироваться в соответствии с требуемыми изменениями (см. рис. 12.1).

Важно понимать эту базовую модель гибких возможностей, которые стараются отвечать потребностям приложения, когда вы рассматриваете переход к облаку, а также грамотно прогнозировать, как преимущества этого перехода скажутся на вашей архитектуре.

Рассуждения об облачной архитектуре

Учитывая предыдущие рассуждения, можно понять, что монолитные приложения предоставляют мало возможностей для того, чтобы можно было соответствовать такому уровню изменяемости. Если у какого-либо компонента приложения возрастает нагрузка, то все приложение придется дублировать, и понадобится создавать новый экземпляр. Представьте, что у вас есть несколько каналов для осуществления бизнеса, и только на один из них по выходным приходится нагрузка (пусть, например, это канал, связанный с интернетом). В таком случае вы захотите масштабировать работу только этого канала, а другие пусть работают, потребляя минимальные ресурсы. Если все приложение исполнено как монолит, вам придется обеспечить работу всех каналов, которые будут потреблять одинаковое количество ресурсов, а это обойдется намного дороже.

Но если архитектура позволяет независимо масштабировать отдельные сервисы (как в случае с микросервисами – см. главу об архитектуре), то масштаби-

рование можно проводить более избирательно, а значит, будет легче оптимизировать затраты. Это основное преимущество облака.

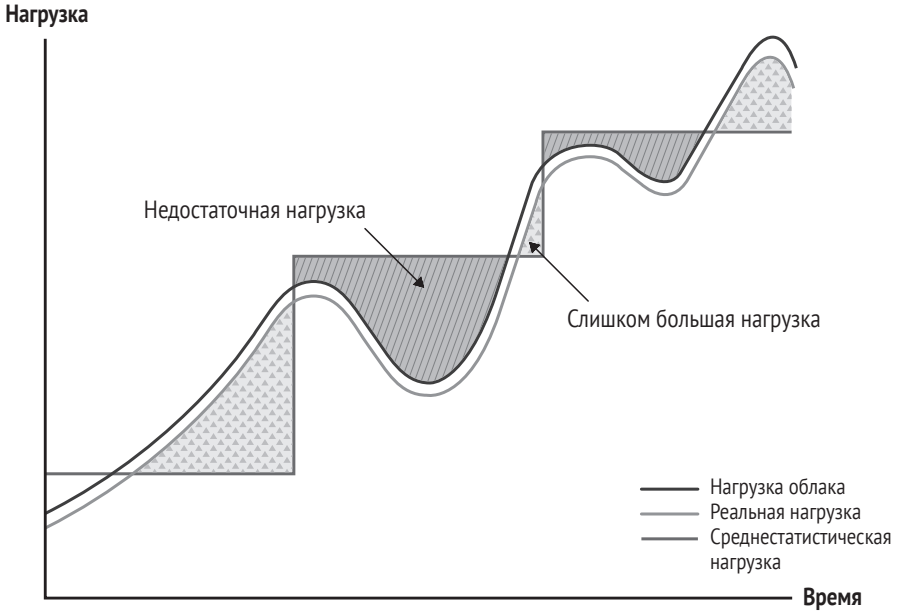


Рис. 12.1. Нагрузка и время:
если все сделать правильно, то нагрузка облака
растет соответственно потребностям

Тем не менее чем больше вы разбиваете элементы архитектуры, чтобы воспользоваться ранее описанными преимуществами, тем больше вам придется иметь дело с подвижными частями архитектуры. При избытке подвижных частей придется совершенствовать возможность управления множеством версий. Вместо того чтобы управлять двадцатью приложениями, вам, возможно, придется управлять 150 сервисами, у каждого из которых к тому же свой собственный ряд версий. Это почти нереально, если не обеспечить достаточный уровень автоматизации – а это подразумевает все, что нужно для автоматизации модели доставки с применением облака.

Другая проблема работы с таким количеством подвижных частей – повышение риска сбоев. С одной стороны, вроде бы ничего страшного, но как все-таки уменьшить список того, что может отказать в нашей архитектуре? Концепция архитектуры, которую мы здесь рассматриваем, называется *элегантной деградацией*. Представьте себе интернет-магазин, например Amazon, который может предоставлять вам персонализированные рекомендации, основанные на ваших предыдущих покупках. Если этот сервис по какой-то причине перестанет работать, сайт не будет выдавать ошибку при загрузке или позволять задержкам в получении ответа на запрос отображаться; вместо этого Amazon

будет всего лишь показывать вам некоторую статичную рекомендацию. Возможно, ваше впечатление от работы сервиса будет слегка подпорчено (хотя многие просто ничего не заметят), но это все же лучше, чем получить сообщение об ошибке или об истечении времени ожидания. Идея элегантной деградации работает во многих контекстах, но не во всех, и обычно представляет более экономную альтернативу поддержания доступности всех ресурсов одновременно.

Облачная архитектура часто подразумевает использование пакетов или шаблонов (например, Docker-шаблонов) из облачных репозиториев, что прекрасно сочетается с перспективой повторного использования. Но это не значит, что вам не нужно обслуживать данные зависимости. Например, очень маленький пакет чуть не оказался причиной крупных перебоев доступности интернета в 2016 году, когда разработчик по имени Азер Коцулу взял пакет из интернета, который использовался во многих других системах, и в итоге нечаянно навредил работе этих систем [1]. Другой риск в том, что в используемых пакетах могут быть уязвимости, о которых вы не знаете. Джош Корман говорил об этом в 2015 году на корпоративном DevOps-саммите, ссылаясь на исследования, которые он провел с целью узнать, сколько open-source и сторонних компонентов находится среди шаблонов Docker (многие из них оказались неизвестными, совсем как в вышеприведенном примере с малыми пакетами). Корман обнаружил, что более чем у 20 шаблонов Docker имеются известные уязвимости. Это подвергает вас множеству рисков [2]. Поэтому недоступность (или смена версии), а также известные уязвимости, скрывающиеся в шаблонах, – это то, что вам приходится активно обслуживать в вашем приложении.

Один из моих клиентов решился на весьма консервативную стратегию, состоявшую в создании внутреннего репозитория для каждого используемого компонента. Копия каждой используемой версии скачивается и хранится из публичного облака для каждой из используемых версий, а также активно обслуживаются эти версии. Новые версии не начинают использоваться ими автоматически, им нужно потратить силы, для того чтобы посмотреть, нет ли новых версий, а затем нужно взять их для себя. Но это значит, что они могут предотвратить случаи недоступности и конфликтов версий, а еще у них будет непосредственная возможность проводить анализ на наличие известных уязвимостей. По стоимости это не всегда выгодно, но вы можете оценивать риск, выбирая, насколько активно вы управляете этой частью облачной архитектуры.

Управление облаком

Гибкость, которую обеспечивает облако, – это замечательно. Типичное узкое место проектов (из-за недостатка окружений) исчезает, и освобождаются ресурсы для доставки. Но гибкость таит в себе и новые проблемы. Раньше для выделения средств на развитие вашей инфраструктуры требовалось проведе-

ние детального планирования, и к делу подходили со всей внимательностью. Но в случае с облаком любой разработчик, имея на руках корпоративную кредитную карту, может приобрести услуги облачных сервисов. Один из клиентов, с которым я работал, открыл публичную облачную инфраструктуру для его разработчиков, и бюджет на окружение, рассчитанный на полгода, исчерпался в первый же месяц. Само собой, это послужило хорошим уроком для организации.

В то же время, если от длительного процесса закупок для инфраструктуры вы придете к длительному процессу одобрения использования облачной инфраструктуры, такой переход не даст преимуществ. Главное здесь – найти золотую середину. Вам придется сформировать новую модель управления, которая будет учитывать стоимость и скорость работы, при которых команды смогут принимать правильные решения в процессе обеспечения окружений.

Логическое управление окружением все еще остается обязательным при работе с облаком. При работе с моделью обслуживания локального сервера основная функция заключается в том, чтобы контролировать то, кто может пользоваться окружением и в какой момент, а также то, какая конфигурация используется в данной среде в определенный момент времени. В работе с облаком вам также хочется убедиться в соблюдении определенных стандартов вашей организации и в том, что для этого предоставляются нужные шаблоны. В дополнение нужно убедиться, что у вас есть стратегия использования отдельных окружений и место, в котором вы сможете собрать и протестировать все приложения. В конце концов, вам нужно точно знать, что есть место для тестирования конфигураций приложений, которые будут использоваться в продуктивной среде. Разветвление окружений и связанных с ними конфигураций приложений может значительно усложнить задачу, если никто не будет этого контролировать.

Организации могут как-то справиться с некоторыми из этих задач, пользуясь системой управления облачными ресурсами или системой-посредником, которая будет соответствовать контексту вашей организации в консоли управления. Такая система также обеспечивает следование стандартам безопасности, когда поставляются только одобренные к применению шаблоны и предусмотрены ограничения бюджета. В идеальном случае такая система может определить, какой из поставщиков облачных сервисов может предложить самую низкую цену, и предоставить окружение этого поставщика, но по мере написания этой книги я все больше убеждаюсь, что ситуация смены поставщиков менее реалистична, чем мы думаем. Каждый поставщик предлагает специфичные сервисы, предоставляющие дополнительные преимущества, но вместе с тем до определенной степени закрепляющие вас за ним. Управление такими системами-посредниками, однако же, принесет немало пользы. Наличие облака окажется кстати, если у вас множество подвижных компонентов, и чем больше их будет появляться, тем лучше должны быть подходы к управлению ими. Спросите себя, насколько хорошо вы управляете окружениями и как много их у вас. Теперь представьте, что у вас облачная архитектура, в которой

вы сосредоточили в десятки или сотни больше окружений. Будет ли текущий подход к управлению окружениями по-прежнему работать?

Проектирование надежности сайтов

При работе с облаком традиционные концепции администрирования также необходимо пересмотреть. Понятие, которое зачастую связано с этим переходом, – «проектирование надежности сайтов» (Site reliability engineering, SRE), популяризированное Google.

Исходя из предположения о том, что облачная архитектура работает с малыми компонентами, что прекрасно сочетается с необходимостью сокращать ресурсы, вы вряд ли встретитесь с полной недоступностью ваших ресурсов (хотя для этого сам поставщик должен быть полностью доступен). Это, в свою очередь, означает, что вместо традиционного показателя периода доступности

$$\frac{\text{период недоступности компонента}}{\text{весь период}}$$

(который имеет смысл использовать, когда вы наблюдаете за малыми компонентами) вам стоит предпочесть показатель доступности сервиса (например, определенный сервис может поставляться со специфичными ему производительностью и качеством):

$$\frac{\text{всего запросов} - \text{неудачные запросы}}{\text{всего запросов.}}$$

В управлении этими показателями доступности используется понятие о *суммарной погрешности*. Идея в том, что у каждого сервиса есть стоимость периодов недоступности сервиса по причине запланированной деятельности (например, обновления или развертывания) или некоторой проблемы. Имея этот показатель, команды могут в определенной степени решать, что с ним делать (например, если они уверены в том, что развертывание потребляет 5 % бюджета, но расчеты неверны и на самом деле потребляется 25 % ресурсов, то у них будет меньше ресурсов для выполнения других задач). Это мотивирует команды находить менее затратные решения для обновлений и развертываний, а также более подходящие способы поддержания доступности сервисов. Также наличие общего для Dev- и Ops-команд показателя часто используется в кругах DevOps для согласования работы команд. Облачные решения нередко разрабатываются обеими командами, чтобы можно было минимизировать воздействие на эту стоимость.

Одним из релевантных показателей, который поможет вам свести к минимуму ошибки при расходовании бюджета, является *время до восстановления* (mean time to recovery, MTTR). Его часть называют ключевым показателем. Он охватывает время, за которое восстанавливается доступность сервиса с мо-

мента обнаружения проблемы. Вместе с показателем *времени до обнаружения* (mean time to discovery, MTTD), который охватывает период с момента возникновения проблемы до ее обнаружения, эти метрики помогут вам справляться с незапланированной недоступностью. Учитывайте эти две метрики в ваших оценочных таблицах DevOps или SRE – вместе с метрикой доступности, – чтобы все могли их увидеть и вы смогли оценивать и измерять ваше совершенствование со временем.

Мониторинг в SRE-модели (безусловно) должен разделяться на три категории:

1) уведомления: это когда вам нужно решительно что-то делать, чтобы восстановить работу сервиса. Помните, что мы не просто пытаемся здесь успеть восстановить все как можно раньше, и мы не будем будить инженера, если сервер или сервис не работает, но это не окажет критического воздействия на пользователя. Нам нужно придерживать решение этого вопроса до подходящего момента. Иначе наши дежурные инженеры вскоре перегорят;
2) тикеты: существуют проблемы, которые мы обнаруживаем, но не нуждаемся в их срочном разрешении. При мониторинге системы должны создаваться тикеты, которые сразу будут приоритизироваться в бэклоге и обрабатываться в нормальное рабочее время;
3) логи: они окажутся полезными не только при возникновении проблем, но еще и когда нужно будет проводить анализ тенденций и определять области, нуждающиеся в улучшении.

Проектирование надежных сайтов формализует использование научного метода для постоянного совершенствования. Поэтому вам нужно каждый раз прогнозировать, к чему приведет то или иное улучшение, оценивать исходные условия, а после реализации проверять, действительно ли улучшение принесло пользу.

Тем не менее важно стремиться к лучшему, всячески мотивируя к этому вашу команду и не подразумевая некого наказания за то, что улучшения не были сделаны раньше. Тактика подведения итогов без поиска виноватых символизирует переход от поиска корня проблемы, когда организация пытается выявить команду или человека, ставшего причиной возникновения проблемы, к прогрессивной культуре, в которой проблема используется для выявления способов улучшить систему и по возможности исключить повторение той же ошибки в будущем. Как я отмечал в главе 8, Etsy придерживается идеи о том, что каждый новый сотрудник должен сразу начинать совершать развертывание в продуктивной среде. Вы могли бы подумать, что это рискованно, но в Etsy считают, что если новичок может что-то сломать в продуктивной среде, это всего лишь сигнал: система недостаточно прочна, чтобы выявлять простые проблемы, которые может привнести новичок. Сосредоточенность на совершенствовании системы, которое ставит целью упростить ее применение, аналогичным образом используется в сообществах DevOps и SRE.

Этой перемене культуры способствует то, что DevOps-практики и облако помогают справляться с изменениями, вовлекая меньше рисков. Так или иначе, развертывание в продуктивную среду можно рассматривать как очередную стадию тестирования. Если вы развертываете всего лишь небольшой пакет изменений в продуктивную среду и тестируете новую версию при помощи небольшого потока данных из продуктивной среды – это можно назвать *канареечным тестированием* (дело в том, что когда-то канареек использовали в угольных шахтах для выявления мест скопления газа), – то любое отрицательное воздействие будет ограничиваться количеством данных, которое мы передали новой версии. Мы можем регулировать тестирование согласно нашим требованиям по рискам, а также откатывать изменения при необходимости.

Та же техническая конфигурация, что и в случае с канареечным тестированием, позволяет нам проводить и A/B-тестирование; таким образом, мы можем использовать две различные конфигурации сервиса в продуктивной среде и смотреть, какая из них показывает наилучшие результаты, а затем, когда мы уже уверены в том, что нашли наилучшую альтернативу, мы выкатываем эту конфигурацию полностью на продуктивную среду.

Я хотел бы упомянуть еще два признака хорошей облачной архитектуры и организаций, которые их достойно обслуживают. Во-первых, это предсказуемость: поставщики облака позволяют вам осуществлять динамическое масштабирование в ряде ваших сервисов. Различные обстоятельства могут все же вынудить вас произвести некоторое прогнозирование (и вам нужно будет проводить его постоянно). В Австралии проводятся скачки под названием Кубок Мельбурна, на которых формируется фонд в 140 миллионов австралийских долларов на ставках в забеге, который длится секунды и вызывает всплеск интернет-пари [3]. Этот всплеск запросов на сайте Кубка Мельбурна не то, что DevOps-команда сайта может сбросить на плечи поставщика облака: здесь им нужно формировать свое собственное решение для прогнозирования и масштабирования. Хорошие организации понимают, в каком случае условия будут отличаться и какая архитектура будет подходящей для таких случаев, в то время как другие организации просто будут полагаться на стандартные средства, предоставляемые поставщиком облака.

Во-вторых, вам нужно моделировать работу над возможными проблемами. Вы можете использовать нечто вроде «обезьяньей армии» Netflix – в состав которой, помимо других инструментов, входят Chaos Monkey (произвольно отключающий серверы), Chaos Gorilla (отключающий целые зоны доступности Amazon) и Latency Monkey (искусственно провоцирующий задержки во взаимодействии с клиентским сервером), – чтобы посмотреть, способны ли вы устранить такого рода проблемы [4]. Эти инструменты можно использовать регулярно, и они все чаще становятся составной частью стратегии организации для поддержания надежности сервисов. Еще одна стратегия – симуляция крупной катастрофы (такой как землетрясение или пожар; некоторые даже учитывают сценарии вторжения пришельцев), для того чтобы протестировать устойчивость и выявить слабые места. На самом деле эти эксперименты нуж-

ны не для того, чтобы доказать, что вы готовы к катастрофе, а для того, чтобы выявить пути дальнейшего развития. Чем лучше становится ваша система, тем больше усилий нужно приложить для того, чтобы ее сломать. Слабая точка всегда будет, и ваша задача – ее найти. Хорошие облачные архитектуры устойчивы к деструктивному воздействию и улучшаются после каждой попытки их сломать. Архитектуры, которые не тестировались на устойчивость, рискуют сломаться тогда, когда это действительно важно, – во время реального использования. Коротко говоря, хорошая обычная архитектура использует то, что предоставляет платформа и расширяется самой организацией дополнительными возможностями, для того чтобы она могла обеспечить потребности бизнеса и архитектуры приложения.

Первые шаги вашей организации

Пересмотрите ваши облачные приложения

Исходя из того, какие преимущества предоставляет для вас облако (с учетом двух факторов – разветвленности архитектуры и зрелости приложения относительно DevOps-практик), пересмотрите ваши текущие облачные приложения (или те, которые планируете убрать). Чтобы это осуществить, сначала проанализируйте архитектуру для выявления компонентов, которые не зависят друг от друга и позволяют масштабировать их по отдельности. Также выявите сервисы, которые нужно изолировать для целей дальнейшего рефакторинга архитектуры. Пересмотрите каждый изолированный компонент на предмет зрелости относительно DevOps-практик (SCM, управление сборкой и развертыванием, автоматизация тестирования), чтобы выявить бреши, которые необходимо заполнить.

Далее спросите себя, действительно ли вы получите выгоду от гибкости облака, предоставляемого для этих приложений, а также от гибкости архитектуры. Только при наличии подходящей архитектуры и возможностей автоматизации вы сумеете полноценно использовать преимущества облака. Вам стоит начать с обеспечения этих возможностей – перед переходом в облако или по факту перехода, – чтобы сократить стоимость содержания облачной архитектуры, а также риски отрицательного появления воздействия на бизнес в случае обнаружения проблем.

Основываясь на этом анализе, вы получите список приложений, которые уже готовы к переходу в облако, и бэклог предстоящей работы, которая предполагает подготовку все большего количества приложений для перемещения в облако посредством рефакторинга архитектуры и обеспечения дополнительных возможностей DevOps.

Прорепетируйте ситуацию катастрофы в облаке

Выберите какой-либо сценарий для тестирования (например, ваш поставщик облака обанкротился, и вы потеряли доступ ко всем системам и дан-

ным, хранящимся в облаке) и проведите несколько экспериментов: что вы будете предпринимать, чтобы все восстановить. Это будет подразумевать, например, создание новой инфраструктуры у разных поставщиков облака, установку приложений, необходимых для обслуживания бизнеса, а также восстановление данных из внешнего источника. Здесь нужно сделать две вещи:

1) выявить слабые места и приоритизировать их для того, чтобы усовершенствовать вашу облачную архитектуру;
2) оценить область поражения и затраты времени на восстановление, чтобы понять, что можно было бы улучшить в дальнейшем.

ЗАКЛЮЧЕНИЕ

Осознанная работа

> Я убежден, что самообучение – это единственно возможный вид обучения.
>
> *Айзек Азимов, «Прошлое науки – будущее науки»*

На протяжении всей книги мы в деталях обсудили, насколько должна трансформироваться работа организации, для того чтобы можно было отойти от производственного стиля мышления. Такую перемену осуществить нелегко, и не все методики из тех, что я упоминал, подходят каждой организации. Более того, важно, чтобы вы вкладывали время, силы и средства в правильные преобразования, иначе вы еще больше запутаетесь. Тем не менее потенциальная выгода очевидна, и я советую вам при любой возможности выискивать способы реализации современных технических подходов в вашей компании. Работа ИТ – креативная работа, для выполнения которой требуются творчески мыслящие люди, а не роботы, механически делающие одно и то же. Будучи руководителями и менеджерами креативно настроенных сотрудников, мы должны оставаться в курсе того, что происходит в индустрии, чтобы развивать наши рабочие процессы в верном направлении. В конце концов, нам нужно использовать все новые технологии и практики, способные укрепить бизнес. Необходимо постоянно учиться.

Теперь, когда мы достигли конца книги, я хочу поделиться некоторыми соображениями по поводу того, как удовлетворять потребность в непрерывном обучении, а также поделюсь с вами ресурсами, которые использую для пополнения своих знаний. Эта информация пригодится тем, кто считает себя сознательным работником, особенно тем, кто работает в ИТ.

Как управлять собой в мире, где ваша работа меняется каждые несколько лет и новые инструменты и технологии изобретаются быстрее, чем когда-либо? Работа в современной ИТ-организации может пугать: многое меняется на

глазах, традиционные роли смещаются. Agile, в частности, стал испытанием для многих проектных менеджеров, поддерживающих проектные планы, составленные задолго до этого. А новые техники и инструменты продолжают появляться все быстрее.

Вот что я уяснил за последние несколько лет: организации не смогут обучить вас всему, что вам нужно. Я считаю, что в Accenture самая лучшая программа обучения в мире, она охватывает почти все темы, но вам нужно не только развивать широкий ряд навыков, но еще и продвигаться вглубь, осваивая каждый из них. Для этого обычно нужно больше, чем компания может обеспечить, и для погружения организации в совершенно новую тему чаще всего требуется время. А посему где можно пройти дополнительное обучение и чего это будет стоить?

- Конференции: хотя я понял, что невозможно составить бизнес-кейс для конференций, ибо большая часть того, что вы узнаете, принесет результат только через несколько месяцев, это все еще наилучший источник новых тенденций и идей. Если вы подойдете к кому-то из спикеров после выступления, то узнаете чуть больше о неудачах и горьком опыте на его пути. Постарайтесь поговорить с людьми откровенно, чтобы сделать для себя полезные выводы.
- Локальные митапы: похожее занятие – посещение локальных митапов. Сейчас проводится такое количество митапов, что бывает сложно выбрать наилучший из них. Мой совет: выбирайте те, которые вписываются в ваш график. В моей организации я поручил одному человеку проводить регулярные собрания для группы сотрудников: так им легче посещать митапы. Таким образом можно объединить тимбилдинг и обучение чему-то новому. Это бесплатно! Я имею в виду, насколько лучшим может казаться такой тимбилдинг для компании? Возможно, вам стоит периодически объявлять и чествовать в вашей организации чемпиона по посещению митапов.
- Массовые открытые онлайн-курсы, такие как Coursera, бесплатно предоставляют слушателям курсы университетского уровня. Они обычно выстраиваются согласно материалам видео- и аудиолекций и дополняются заданиями, которые необходимо выполнить для закрепления пройденного (для этого понадобится не только смотреть видео, но и читать тексты, что может отнимать довольно много времени). Я проходил несколько десятков таких курсов; некоторые из них были связаны с работой (искусственный интеллект, программирование, блокчейн), другие затрагивали общие темы, такие как искусство и политика. Это замечательный способ узнавать что-то новое, если вы можете позволить себе потратить на это пару часов в неделю.
- Open-source проекты: если у вас есть несколько свободных часов, но вы не хотите потратить их на структурированную деятельность, например прохождение онлайн-курсов, то можете предпочесть open-source проект,

что позволит вам осваивать новые технологии и инструменты. Лично я не занимался этим продолжительное время, я занимался лишь короткими домашними проектами, где писал небольшие программы для упорядочивания моей фотоколлекции или для сбора данных из различных источников для оптимизации планирования моего бюджета. Я уверен, что есть много способов быстро добиваться результатов, но обучение, как ни крути, важнее, чем выполнение текущего задания.

- Блоги и подкасты: если у вас не очень много времени, то блоги и подкасты, возможно, подойдут вам больше. Вы можете следить за частными блогерами, такими как я, или воспользоваться агрегаторами блогов, такими как Dzone, InfoQ, DevOps.com, либо DevOps-блогом Accenture, где представлены проработанные статьи блогов, охватывающих широкий круг тем. Подкасты – также хороший источник информации: их можно слушать по дороге на работу, занимаясь спортом или путешествуя. Таким образом, вы сумеете уделить время самообразованию даже в условиях, когда это обычно полагают невозможным. Можно было бы, как я или многие другие, вместо этого зависать в интернете или тысячу раз проверять почту… Но, согласитесь, изучать что-то новое во сто раз полезней!
- Книги: и конечно же, всегда найдется книга, которая вас чему-то научит. В конце этой книги приводится список литературы, которую я настоятельно рекомендую к прочтению.

Тайм-менеджмент

Быть новоиспеченным родителем, писать эту книгу (а также вести блог до и после ее выхода), работать консультантом в международной компании и при всем этом поддерживать здоровый образ жизни – для меня испытание не из легких. Мир технологий быстро развивается, и если погружаться в него лишь на работе, со временем вы потеряете связь с последними тенденциями. Для поддержания статуса ответственного работника вам нужно быть в курсе происходящего и находить время на изучение новостей. Все, что касается вашего саморазвития, крайне важно, и если вы не отводите для этого достаточно времени, то не сможете поспевать за прогрессом. Как я уже говорил, проблемы, которые существуют на данный момент в индустрии, на мой взгляд, частично вызваны нами самими, потому что как лидеры мы должны держать руку на пульсе и следить за переменами. Мы слишком сильно полагаемся на расхваленные подходы, продолжая действовать по шаблону, вместо того чтобы попытаться понять, как последние тенденции формируют новый мир ИТ и бизнеса.

Для меня каждая тренировка – это возможность послушать подкаст (пусть не всегда, но довольно часто я сопротивляюсь соблазну послушать музыку и предпочитаю выбрать подкаст). То же самое касается походов на работу и путе-

шествий. Я часто добираюсь до работы на велосипеде и слушаю подкаст – получается, что я одновременно выполняю три задачи: тренируюсь, еду на работу и узнаю что-то новое.

На работе много отвлекающих факторов, и порой твой календарь забит под завязку всякого рода совещаниями, важными и не очень. Один из способов сберечь полезные рабочие часы – распланировать время на работу и на проверку почты. Если у вас есть дела, которые должны выполняться регулярно (например, подготовка текущего отчета, написание статьи для блога или составление отчета по расходам), пометьте их в вашем календаре. То же касается и почты: вместо того чтобы просматривать ее когда придется, зафиксируйте это занятие в определенном получасовом временном промежутке дня. И наконец, выделяйте интервалы времени на вашу основную работу – когда вы будете создавать продукт, писать код или просматривать чью-то работу. Если в вашем календаре есть наиболее приоритетные пункты, то другие несрочные дела должны занимать только пустые места в этом календаре. Так вы не обнаружите себя составляющим отчеты по ночам из-за того, что время длиной в день, неделю или месяц умудрилось ускользнуть от вас.

В этой книге я не стремлюсь дать вам пошаговые инструкции: я не тешу себя мыслью, что вы в точности будете повторять то, что я вам расскажу (как консультант и как муж я знаю: к тому, что твои ценные советы игнорируют, поневоле привыкаешь). На что я действительно надеюсь, так это на то, что вы позаимствуете из книги некоторые идеи. Все, что я написал, выстрадано на опыте. Как и все вы, я продолжаю учиться и развиваю свои подходы к каждому проекту, клиенту и к сотрудничеству с людьми, практикующими DevOps.

Для того чтобы очистить ИТ-индустрию от накопившихся стереотипов, как я уже говорил, нужно много усилий. Новые технологии и практики будут появляться непрерывно. Эта книга, мне кажется, будет актуальна еще долгое время, но мы продолжим развиваться, и следующее поколение лидеров будет воспринимать новый на сегодняшний день образ мышления как данность. Подготовительные траты будут по-прежнему сокращаться (просто подумайте, например, о функциях Amazon Lambda), и вместе с этим меньшие партии работы и ускоренная доставка станут реалистичными. Нам нужны новый образ мышления и новые техники управления, чтобы справляться со сложностью, которая возникает из-за таких подвижных явлений. Но сначала давайте сосредоточимся на нескольких шагах в ближайшей перспективе и с них начнем развивать наше собственное решение. В книге я представил свои проверенные рекомендации и поделился опытом работы с десятками проектов и компаний. Надеюсь, это поможет вам на пути к трансформации вашей организации. Я видел, что происходит, когда организации совершенствуются и люди, которые работают в ИТ, по-настоящему увлекаются своим делом. Наблюдать такие изменения – лучшее вознаграждение, и все это очень помогает бизнесу. Вам не нужно становиться еще одним Netflix или наилучшим ИТ-отделом в стране, чтобы оказывать влияние на своих сотрудников и бизнес. Продолжайте движение, и кто знает – может быть, вы станете примером для подражания.

Подозреваю, что у вас, как и у всех нас, свободного времени не в избытке. К тому же, потратив деньги на эту книгу, вы часть времени посвятили ей, а это для меня еще более ценно. Спасибо вам! Эта книга – плод многих размышлений, призванных улучшить индустрию, в которой мы работаем. Я буду рад, если вы напишете мне, что сработало для вас, а что – нет: это поможет мне продолжить свои изыскания. Не стесняйтесь высказывать свои соображения и пожелания. Возможно, мы могли бы даже встретиться на одной из конференций или на локальном митапе. Если увидите меня, подходите поболтать!

ПРИЛОЖЕНИЕ

Аналогия с заводом: подробности

Фундаментальный принцип: процессы в производстве и креативные процессы в ИТ

Концепция производства состоит в том, что мы проектируем продукт, определяем процессы производства, а затем производим ряд одинаковых товаров. В ИТ мы поставляем решение, которое уникальным образом воплощается в нашем контексте каждый раз, когда вносятся изменения. Мы никогда не делаем один и тот же продукт, используя все те же компоненты и такой же исходный код в точно такой же архитектуре. Устаревшие процессы производства стремятся к уменьшению вариативности. В ИТ мы стремимся при помощи вариативности находить наилучшее решение проблемы, тем самым удовлетворяя наших клиентов.

Оценка продуктивности и качества на основе стандартизированных результатов

Мы уже говорили о том, что в ИТ одно и то же решение никогда не поставляется дважды, между тем как в производстве поставляется большое количество одинаковых товаров. Это, в свою очередь, влияет на оценку продуктивности и качества. Давайте начнем с чего-то менее противоречивого – с качества.

В производстве, если мы изготовляем одинаковые товары, оценить качество легко, если у нас есть некий образец или спецификация. Любое отличие от образца будет восприниматься как снижение качества, и мы можем расценивать количество таких расхождений как меру качества нашей производственной системы. При наличии системных проблем, если исправить их, решение распространится на все копии продукта. Тестирование произведенного продукта часто осуществляется стохастически – мы выбираем некоторое количество образцов из нашей производственной системы, чтобы удостовериться, что вариативность в производстве остается в пределах ожидаемых показателей.

В ИТ у нас нет «правильных» образцов или целевых спецификаций. Мы осуществляем тестирование согласно требованиям, пользовательским историям или спецификациям проекта, но исследования и опыт показывают, что в них кроются источники многих дефектов в ИТ. Поэтому на образец или целевую спецификацию не получится полностью положиться. Дефекты чаще всего исправляются путем решения проблемы отдельного товара, а не всей производственной системы. В итоге количество найденных дефектов не всегда говорит о самом уровне развития системы производства, так как мы можем продолжать поставлять все тот же неэффективный код, что и раньше. Оценивать качество в ИТ нужно по-другому. (Как это делать, было показано в главе 7.)

Продуктивность в ИТ оценивать еще сложнее[1]. По правде говоря, я был участником многих дискуссий, круглых столов и разговоров о метриках, но пока что выверенного средства для оценки ИТ-продуктивности не нашел никто из тех, с кем я общался. Многие CIO признавались, что им не нравится то, что они используют для оценки продуктивности. Если представить это все в контексте отличия ИТ от производства – то, что в ИТ креативные начинания позволяют находить уникальные решения, и как это выглядит в сравнении с массовым производством, – то становится очевидно, что добиться правильной оценки продуктивности сложно. Как бы вы измеряли продуктивность работы отдела маркетинга, писателя или автора песен? Вы можете оценить результаты (за год стало больше листовок, больше книг, больше песен), но лучше ли от этого стала работа отдела маркетинга, писателя и автора песен?

Думаю, вы согласитесь, что результаты (например, успешная маркетинговая кампания, продажи бестселлера или популярность песни) более важны, чем конкретный продукт вашей работы. Раньше мы измеряли продуктивность количеством строчек кода, *функциональными точками* и другими количественными показателями, хотя мы довольно быстро найдем аргументы против того, чтобы их использовать. С приходом Agile мы стали использовать стори-поинты и высчитывать скорость работы по проекту (velocity), которые являются шагами для нахождения хорошего ответа на наш вопрос, так как с их помощью можно оценить поставляемую функциональность. Но по своей сути продук-

[1] В одной из статей в своем блоге я рассуждал о том, что продуктивность – это показатель, который сложно измерить в ИТ. Вместо этого вы можете измерять продолжительность циклов, расходы и поставленную функциональность [1].

тивность все еще остается показателем, который измерить сложно. Это приводит к развернутым дискуссиям, особенно тогда, когда вы работаете с партнером по ИТ-доставке, которого хотите убедить в том, чтобы он работал более эффективно. (Я раскрываю эту тему подробнее в главе 4.)

Я бы хотел, чтобы кто-нибудь потратил немного денег и поэкспериментировал с запуском параллельных проектов на Agile и водопаде, с распределенными командами или размещенными в одном месте и т. п., чтобы можно было продемонстрировать, какой из вариантов лучше, и доказать всем, кто задерживает доставку своей позицией, благосклонной к водопаду. В то же время среда делает выбор за нас: стремительно развивающийся рынок с постоянно меняющимися требованиями мешает упрямо следовать методологии водопада.

Функциональная специализация и набор навыков сотрудников

На традиционном производстве оттачивать процессы стремятся при помощи высококвалифицированных и узкоспециализированных сотрудников, вместо того чтобы уделить внимание тем сотрудникам, которые являются неотъемлемой частью процесса. Методологии ИТ-доставки изначально следовали такому подходу, поскольку применение хорошо структурированной методологии несет в себе ряд преимуществ. ИТ сегодня полагается на креативных сознательных сотрудников, что отличает эту индустрию от традиционного производства. Чрезмерно узкая специализация сотрудника, который выполняет одну специфичную задачу в цепочке доставки ценности в ИТ (например, тестирование или разработку), доказала свою неэффективность, так как узкоспециализированным организациям бывает трудно сложить общую картину контекста. Учитывая, что решения в ИТ не повторяются, освоение контекста – это очень важный этап обеспечения успешной доставки.

Рис. П.1. Т-образная схема навыков:
сотрудники, обладающие Т-образными навыками,
имеют больше навыков, чем «I-образные» сотрудники

Движения Agile и DevOps пытаются преодолеть разобщенность, которая присутствует на протяжении всего цикла разработки продукта, и преследуют цель сделать работу организации более ориентированной на контекст, более быстрой и высококвалифицированной. Вместе с этим на смену идеальному работнику, работающему только в одной области, приходит человек с навыками в более широком ряде областей. Сотрудники с T-образными навыками имеют хорошее понимание нескольких областей и глубоко специализируются в одной из них, между тем как сотрудники с I-образными навыками обладают знаниями лишь в одной области [2]. Идея, стоящая за всей этой специализацией, в производстве должна была позволить менее опытным работникам научиться хорошо выполнять работу, полагаясь на исчерпывающее количество инструкций для процесса производства (которые были составлены квалифицированным инженером), что помогало в некотором роде восполнить нехватку навыка. Такой вариант, к сожалению, невозможно осуществить в ИТ из-за креативной и сложной природы работы.

Предсказуемость процесса производства и управление им

Процесс традиционного производства довольно жестко детерминирован. Как только вы определитесь с процессом производства, необходимыми ресурсами, вы начнете получать ожидаемый продукт. К сожалению, ИТ это не свойственно. Следование методологии, которая успешно сработала в другом проекте, не гарантирует того, что в новом проекте вы получите аналогичные результаты. Конечно, существует определенная корреляция с тем, что организации становятся более успешными: в результате некоторые методологии применяются более широко, чем другие (например, SCRUM или PMBOK). Но это методология, которая не настолько эффективна для ИТ, как для традиционного производства.

Умение предсказывать результаты производства предоставляет возможность исправлять проблемы с продуктом, изменяя уже сам процесс. Это не случай ИТ: просто внести изменения в процесс – не значит исправить его. Многие профессионалы по смене подходов к управлению в этом убедились.

Более того, наличие множества ресурсов для креативной работы подразумевает, что процесс сам по себе будет более сложным и менее предсказуемым. Процесс управления, который подразумевает такой же уровень предсказуемости, как на производстве, в итоге приводит к появлению массы непродуктивных явлений в организации. Сотрудники будут стараться подгонять цифры под ожидаемые, но это осуществляется в рамках процесса реагирования на непредвиденные ситуации или при помощи манипулирования данными результатов работы процесса и с использованием всякого рода неоднозначных моментов в ИТ. Более эмпирический подход, такой как Agile, позволяет нам показывать действительный, менее точно прогнозируемый прогресс и приводить наши ожидания в соответствие ему. Дон Райнертсен хорошо описал слу-

чаи, когда предположение о том, что доставка ИТ-продукта может быть предсказуемой, может принести немало проблем, связанных с менеджментом. Он объяснил, что традиционное производство основывается на повторяющихся и предсказуемых задачах, в то время как доставка продукта – по своей сути процесс, уникальный в каждом случае. Попытки использовать техники, которые работают для предсказуемого процесса доставки, в стремительно меняющейся среде приведут к неудаче [3].

Подробнее аспекты управления освещаются в главе 3. А здесь лишь замечу, что если вы, рассматривая графики burndown или burnup (Agile-механизмы для составления отчетности о работе в проекте), видите полное соответствие действительных значений планируемым, то вы, скорее всего, где-то смухлевали.

Важность предварительного планирования и возможность рассчитывать на него

Благодаря тому факту, что процессы производства более предсказуемы, чем издержки на подготовку, стоит полагать, что есть смысл выделять больше времени и сил на предварительное планирование. Это означает, что мы можем составить план изготовления нового продукта на производстве, сделать его прототип, а затем выпускать один и тот же продукт партия за партией.

В ИТ каждый из продуктов уникален, как мы говорили ранее. Это значит, что мы не сможем добиться по-настоящему производственного процесса, но вместо этого будем работать в среде, которую скорее можно сравнить с процессом прототипирования в производственной среде. Однако мы часто пытаемся использовать опыт процесса производства, подразумевающего все это планирование, вместо того чтобы обратиться к более инкрементальному процессу прототипирования. В итоге мы, получается, ожидаем увидеть предсказуемость там, где существует вариативность результатов.

Как писали Гэри Грувер и Томми Маусер в книге *Leading the Transformation: Applying Agile and DevOps Principles at Scale*, «исполнительным директорам необходимо понимать, что управлять ПО и процессами планирования так же, как и всем остальным в организации, – не самый эффективный подход. Каждое новое ПО проекта уникально, поэтому здесь уровень неопределенности при планировании еще выше» [4].

Управление доставкой

Традиционное производство долгое время руководствовалось научными подходами в менеджменте, и хотя с течением времени они меняются, они все еще являются основой современного производства. По сути, это значит, что нам нравится управлять компонентами доставки в конце процесса: например, нас вполне устроит, что доставка будет осуществляться по принципу «черного ящика», если конечный продукт соответствует изначальным ожиданиям. По-

скольку спецификации здесь тоже играют немаловажную роль, мы вполне можем полагаться на этот процесс.

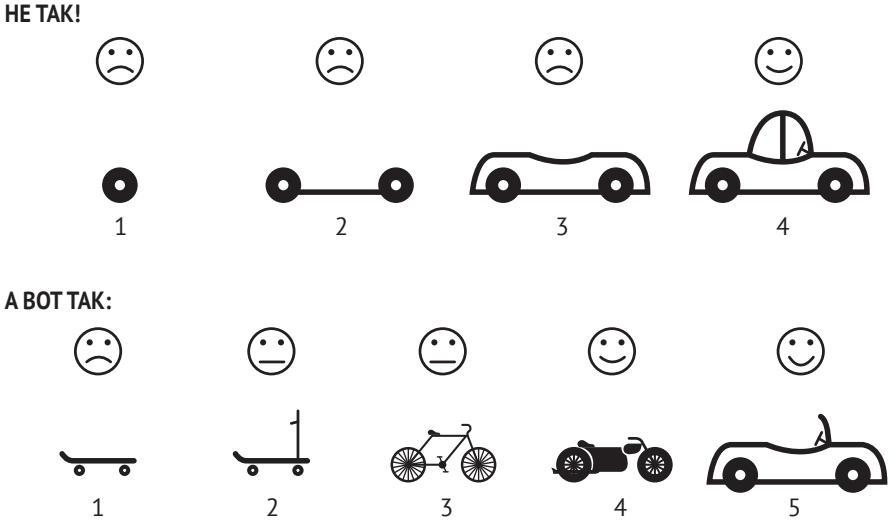


Рис. П.2. Итеративная и инкрементальная доставка:
у итеративной доставки преимущества продукта наращиваются постепенно,
в то время как в инкрементальной продукт можно будет использовать
только после того, как он будет полностью готов

(по рисунку Хенрика Кнайберга «Понимаем MVP (минимальный жизнеспособный продукт)»,
25 января 2016 года, blog.crisp.se/2016/01/25/Henrik_kniberg/making-sense-of-MVP)

В ИТ спецификации не столь однозначные, управление в ИТ нужно осуществлять как можно более прозрачно. С течением времени Agile доказал, что с конусом неопределенности можно обходиться, создавая небольшие инкременты рабочей программы, которые уже можно оценивать при осуществлении каждой итерации. Такая схема не совсем возможна в производстве, поскольку осуществлять множество итераций при создании автомобиля труднее, чем при создании сайта. Поэтому в производстве мы продолжаем полагаться на инкрементальные процессы создания и улучшения. На рис. П.2 наглядно показана разница между инкрементальной и итеративной моделями доставки на примере создания автомобиля.

Автоматизация = продуктивность

Автоматизация – один из аспектов, в котором производство и ИТ-доставка стремятся достичь общей цели: автоматизировать настолько, насколько это возможно. Автоматизация – ключ к продуктивности. В производстве автоматизация подразумевает то, что мы можем изготавливать больше товаров, во-

влекая в процесс все меньше людей; это справедливо и для ИТ. Разница в том, что в производстве автоматизация является частью процесса изготовления (например, автоматизация вносит свой вклад в результаты сборки или создания деталей продукта). В ИТ автоматизация упрощает работу разработчикам и тестировщикам и помогает им быть более продуктивными и креативными, когда выполнение рутинных и малозначимых процессов автоматизируется. На заводах мы часто наблюдаем автоматизацию производства от начала до конца, в то время как в ИТ нам непросто представить такое явление (только если не предположить сценарий доминирования искусственного интеллекта над человеческим). Справедливо для обеих отраслей, что достоверность результатов ухудшается, если для выполнения автоматизируемых задач используются ручные тестировщики. Люди не очень хорошо справляются с повторяющимися задачами, и им стоит сосредоточиться на креативных аспектах работы.

Масштабирование усилий для доставки большей ценности

Все, о чем мы говорили выше, – детерминированный процесс производства, меньшая возможность полагаться на работников, обладающих узкоспециализированными навыками, разница в предсказуемости результатов – указывает на то, что масштабирование производства мы пытались осваивать на протяжении многих лет. Вы строите завод и нанимаете рабочих, и у вас уже есть неплохие шансы на то, чтобы производить какой-либо продукт, даже несмотря на возможные проблемы с культурой или логистикой.

В ИТ при масштабировании усложняются рабочие процессы – они становятся значительно сложнее, чем в традиционном производстве. Здесь присутствует больше заинтересованных сторон, с которыми необходимо взаимодействовать, нужно более широко распространять информацию, и общий контекст нужно составлять, основываясь на взаимодействиях и их границах. Стоимость дополнительного масштабирования в ИТ довольно значительная. Хотя ИТ-системы продолжают расти, нам надо находить способы справляться с этим, помимо того что следует приглашать больше людей. Наши ИТ-системы должны решать крупные проблемы, поэтому следует искать лучшие подходы к масштабированию. Примеры показывают, что если вовлечь чересчур много людей в проект, у которого имеются проблемы, это не улучшит результаты, а только ухудшит их. Фредерик Брукс рассказывал, как приглашение все большего количества программистов в проект IBM не ускорило работу, а еще больше замедлило [5]. Или, как он едко заметил, «вынашивание младенца в утробе занимает девять месяцев, несмотря на количество женщин, причастных к его рождению» [6]. Иметь возможность делать больше, не расширяя круг участников проекта и при этом упрощая поддержку систем, – вот чего добиваются те, кто практикует Agile и DevOps, вместо того чтобы просто раздувать рабочую группу.

Централизация ресурсов

Заводы были механизмом, обеспечивающим концепцию экономии на масштабе. Центральные ресурсы производства, например производственные машины и исходные материалы, хранились в одном месте, чтобы рабочие могли иметь к ним доступ и производить продукт наиболее эффективным путем. В ИТ дела изначально обстояли точно так же. У вас был доступ к мощным компьютерным ресурсам, а позднее и к хорошему интернет-соединению. Сегодня все это вам доступно благодаря широкополосному интернету и облаку. А значит, местоположение определяется не расположением ресурсов, а необходимостью взаимодействовать и иметь возможность пользоваться рядом навыков. Как собрать хорошую команду, чтобы поставлять желаемые результаты? Существуют некоторые ресурсы, которые еще стоит централизовать (например, предоставление удобных инструментов для разработчиков и стандартизированных окружений), но в этом со временем становится все меньше надобности, и это определенно не влияет на выбор местоположения.

Офшоринг

Выведение производства за рубеж было подходящим способом сократить экономическое влияние на производство. Процесс производства можно было воспроизвести за рубежом, и вариативность результатов можно было сносно контролировать. В ИТ слишком часто применяется этот же принцип, но без осознания того, что невероятно важно для успешной ИТ-доставки учитывать контекст – и, соответственно, для получения того же результата необходимо поддерживать каналы взаимодействия. Офшоринг все еще является хорошим способом расширения ваших ИТ-возможностей, особенно когда речь идет о масштабировании. Если говорить об эффективном использовании средств на оплату работы специалистов, то доставка проекта, осуществляемая при помощи штатных команд или команд, распределенных по всему миру, скорее всего, обойдется в ту же цену. Распределенность команд обычно замедляет процессы из-за проблем с коммуникацией; таким образом, проекты с распределенными командами обычно длятся дольше. Во многих случаях штатные команды не способны выступать в роли подходящего варианта из-за требуемых навыков и доступного количества инженеров. И возможностями офшора можно воспользоваться, чтобы успешно поставлять сложные проекты. К сожалению, многие ИТ-директора все еще рассматривают офшоринг как средство сокращения расходов, вместо того чтобы пользоваться им для расширения возможностей, что, в свою очередь, приводит ко множеству проблем с неоднозначной репутацией, создаваемой для офшорной доставки.

Аутсорсинг

Аутсорсинг ИТ-услуг осуществлять намного сложнее, чем аутсорсинг традиционного производства. В производстве вы выводите за рубеж изготовление некоторого специфичного компонента, и пока все соответствует спецификациям – все счастливы. К тому же вам удобно контролировать результаты, так как вы производите конкретный продукт.

Аутсорсинг ИТ осуществлять намного сложнее. Спецификации в ИТ сложнее и в большей степени подвержены изменениям. Мы знаем, что многие проблемы с качеством исходят из самих требований, и ни один проект еще не завершался до того, как одна из заинтересованных сторон не изменила требований. Учитывая то, что мы не можем с легкостью предопределить результат и оценить его заранее, нам нужно быть внимательными к процессам. Аутсорсинг-партнер в ИТ должен предоставлять вам возможность, которой вы не имеете, пользуясь штатными возможностями, будь то навыки или опыт работы, а также соблюдать прозрачность и взаимодействовать с вами, поскольку вам придется вместе работать над сложным процессом доставки. Только тогда, когда обе стороны получают свою выгоду, проект будет успешным. Подумайте о том, что это может означать для вашей коммерческой модели и для стратегии работы с людьми в контексте обеих сторон. Слишком много представителей ИТ-аутсорсинга много обещают и затем довольно быстро остывают или выходят в работе на тонкую грань. Мне повезло, что я работал с прекрасными организациями, представители которых тесно сотрудничали со мной, чтобы выстроить взаимовыгодную структуру для доставки проектов.

Надеюсь, вам стало понятно, что идеи, которые привели нас сюда и ранее делали нас успешными, необходимо переосмысливать и адаптировать к нынешней ситуации. Вместо того чтобы думать о трудоемких моделях традиционного производства, нужно использовать нечто подобное на сильно автоматизированное производство. И я думаю, что движение DevOps будет замечательным катализатором изменений в образе мыслей работников компаний.

МАТЕРИАЛЫ ДЛЯ САМОСТОЯТЕЛЬНОГО ИЗУЧЕНИЯ

Книги

- *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation* Джеза Хамбла и Дэвида Фарли: довольно неплохое руководство о том, как реализовывать непрерывную доставку.
- *Leading the Transformation: Applying Agile and DevOps Principles at Scale* Гэри Грувера и Томми Маусера: прекрасная книга с прагматичными советами о том, как начинать трансформацию в ИТ.
- *The DevOps Handbook: How to Create World-Class Agility, Reliability, and Security in Technology Organizations* Джина Кима, Джеза Хамбла, Патрика Дебуа и Джона Уиллиамса: эта книга предоставляет вам объемное руководство для реализации DevOps-практик.
- *The Effective Manager* Марка Хорстмана: замечательная книга о хорошем менеджменте, который сосредоточен на людях, с которыми вы работаете.
- *The Goal: A Process of Ongoing Improvement* Элияху Голдрата и Джефа Кокса: легкое чтиво о бизнесе, которое познакомит вас с системным мышлением.
- *The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses* Эрика Рая: Эрик описывает, как структурированное экспериментирование позволит вам лучше решать бизнес-проблемы.
- *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win* Джина Кима, Кевина Бера и Джорджа Спаффорда: эта книга легко познакомит вас с концепциями и культурой DevOps.
- *The Principles of Product Development Flow: Second Generation Lean Product Development* Дональда Райнертсена: прекрасная книга, содержащая лучшие рассуждения о партиях доставки.

- *Site Reliability Engineering: How Google Runs Production Systems* Бетси Бейер, Криса Джонса, Дженнифер Петофф и Найала Ричарда Мерфи: узнайте о современных методах администрирования для приложений, вдохновленных опытом Google.

Подкасты и онлайн-ресурсы

- *The Agile Revolution:* австралийский подкаст, посвященный Agile (TheAgileRevolution.com).
- *Arrested DevOps:* в подкасте предоставляется информация о предстоящих конференциях, а также звучат рассуждения на тему DevOps (ArrestedDevOps.com).
- *Career Tools:* полезный подкаст компании Manager Tools, предоставляющий рекомендации каждому на пути его карьеры (www.manager-tools.com/all-podcasts?field_content_domain_tid=5).
- *DevOps Café:* разговорный подкаст обо всем, что связано с DevOps (DevOpsCafe.org).
- *The Economist Radio:* ежедневный подкаст с новостями науки и политики (radio.economist.com).
- *Freakonomics Radio:* подкаст об удивительных взглядах науки на мир вокруг нас (freakonomics.com/archive/).
- *HBR IdeaCast:* бизнес-подкаст издания *Harvard Business Review* с глубоким анализом различных тем (feeds.harvardbusiness.org/harvardbusiness/ideacast).
- *Manager Tools*: замечательное руководство для менеджеров и руководителей (www.manager-tools.com/all-podcasts?field_content_domain_tid=4).
- *The Ship Show* (больше не выпускается, но прежде записанные эпизоды все еще доступны): один из самых ранних DevOps-подкастов (TheShipShow.com).
- *Software Engineering Radio:* профессиональный подкаст на технические темы (www.se-radio.net).
- TED Talks: воодушевляющие разговоры, которые часто раскрывают научные и технологические аспекты (www.ted.com/talks).

ГЛОССАРИЙ

12-факторное приложение: архитектурный концепт, предусматривающий 12 критериев для разработки современных приложений.

CRM-система: система управления взаимоотношениями с клиентами, позволяющая компании последовательно работать со своими клиентами и в дальнейшем выстраивать бизнес-отношения.

Definition of done: практика Agile, согласно которой пользовательская история будет считаться успешно выполненной (см. **Definition of ready**).

Definition of ready: практика Agile, определяющая входной критерий готовности пользовательских историй к работе в следующем спринте (см. **Definition of done**).

ERP-система: система управления ресурсами – практика управления всеми ресурсами для производства и выполнения необходимых процессов организации.

Jenkins: инструмент для реализации непрерывной интеграции.

LeSS: Large-Scale Scrum: метод масштабирования в Agile.

MTTD: примерное время, необходимое для подтверждения обнаружения проблемы.

MTTR: примерное время, необходимое для исправления проблемы.

Open source: модель распространения приложений, основанная на идее безвозмездного пользования и добровольного внесения своего вклада в исходный код приложения.

Perl: скриптовый язык программирования, часто используемый в задачах по автоматизации.

PI-планирование: регулярное крупное мероприятие по планированию, которое является составляющей Scaled Agile-фреймворка и на котором собираются все заинтересованные стороны для осуществления следующего цикла планирования.

PMI (ИУП): Институт управления проектами; проводит тренинги и сертификации для проектных менеджеров.

Scaled Agile-фреймворк (SAFe): популярный фреймворк масштабирования для осуществления Agile-доставки.

Site Reliablity Engineering: современный подход к организации процессов эксплуатации, популяризованный компанией Google.

WSJF: система оценки, по результатам которой вы получаете список задач, приоритизированный по соотношению ценности задачи для бизнеса к стоимости ее реализации. Формула этой оценки основана на принципах SAFe: стоимость задержки / время выполнения.

XaaS: модель предоставления услуг «все как сервис».

Артефакты сборки: файлы, полученные в результате сборки билда, обычно в виде бинарного файла, который можно использовать в развертывании приложения.

Бимодальная ИТ: концептуальная идея того, как новые, более совершенные ИТ-системы разрабатываются в отличной от старых систем манере.

Ввод в эксплуатацию: выпуск нового функционала или новой программы, полностью подготовленных к использованию клиентами.

Версионирование: практика хранения множества версий программы или компонента для дальнейшей возможности отката к любой из версий.

Вертикальное масштабирование: техники масштабирования, при которых добавляются дополнительные вычислительные или другие службы для того же экземпляра приложения вместо распределения рабочих процессов на дополнительные экземпляры.

Вызов без сохранения состояния: техника программирования, при которой службе не требуется знать текущее состояние транзакции.

Вызов с сохранением состояния: техника программирования, при которой служба должна запоминать состояние транзакции для последующего успешного его использования в нескольких транзакциях.

Вызов функции: техника программирования, позволяющая использовать функции других приложений или частей текущего приложения.

Вычислительная среда: окружение приложения, которое позволяет выполнять программы.

Горизонтальное масштабирование: техника масштабирования, в которой дополнительные рабочие процессы распределяются на большее количество систем аналогичного размера и формы, вместо того чтобы обеспечивать наличие дополнительных ресурсов для этих систем.

Двухскоростная модель доставки: модель, поддерживающая две разные скорости доставки, которые будут отличать быстро изменяющиеся системы от медленно изменяющихся.

Декомпозиция систем: техника, позволяющая изменять системы независимо друг от друга и подразумевающая создание интерфейсов, которые будут эффективно работать при смене каждой из систем.

Жизненный цикл доставки ПО (SDLC): данный цикл описывает все действия, необходимые для имплементации требований идей к выпуску продукта в массы.

Изоморфизм ИТ-бизнеса: подход к организации, который подразумевает сближение ИТ и бизнес-функций, что поможет упростить коммуникацию между ними.

Индикатор соотношения стоимости/производительности (CPI): метрика, позволяющая измерять объем произведенной работы за определенную стоимость.

Инструментарий DevOps: набор инструментов, сопутствующий таким практикам DevOps, как управление конфигурациями, автоматическая загрузка и автоматизация тестирования.

ИПЛ (NPS): индекс потребительской лояльности – метрика, определяющая степень удовлетворенности услугами, предоставляемыми организацией или провайдером.

Истинно устаревшее ПО: системы, которые более не обновляются, но продолжают работать, поддерживая бизнес-функционал (см. также **Устаревшее ПО**).

Итоги ознакомления: встреча по окончании фазы ознакомления, во время которой более широкий круг участников информируется о результатах окончания фазы.

Канареечное тестирование: метафора отсылает к канарейкам, использовавшимся в угольных шахтах; при данном подходе подразумевается развертывание в ограниченном ряде сред в продакшн для валидации приложения перед выпуском его в более широкий ряд окружений.

Каскадная модель разработки (Waterfall): модель процесса разработки программного обеспечения, в которой процесс разработки выглядит как поток, последовательно проходящий фазы анализа требований, проектирования, реализации, тестирования, интеграции и поддержки.

Коммерческие программные продукты: преднастроенные ИТ-приложения, способные поддерживать конкретные бизнес-процессы и не требующие осуществления дополнительной конфигурации и программирования.

Компиляторы: средства, которые «переводят» программный код в выполнимые программы.

Конечное состояние архитектуры: предполагаемое конечное состояние архитектуры приложения, которая полностью готова к использованию для бизнеса организации.

Контракты разработки, эксплуатации и переходного периода (DOT): популярная структура контрактов, в которых упоминаются поставщики, создающие решения, поставщики, администрирующие решения, и состояние, в котором организация переносит решение обратно в штатную эксплуатацию.

Конфигурация абстрактной среды: такие переменные, как IP-адреса и названия серверов, должны быть абстрагированными, для того чтобы файлы конфигурации содержали в себе лишь плейсхолдеры вместо конкретных значений.

Концепция бюджета ошибок: бюджет, в котором вместо обычной стоимости закладывается покрытие вероятных неполадок или периоды недоступности, с которыми командам следует справиться, чтобы их работа могла считаться успешной.

Микрослужбы: архитектурная парадигма, пытающаяся идентифицировать наименьший независимый компонент, который можно использовать в качестве самостоятельной службы.

Минимально жизнеспособный продукт (MVP): продукт, предлагающий лишь самые базовые и необходимые функции, готовые к использованию клиентами. Прочие функции добавляются позже с обновлениями.

Минимальный жизнеспособный кластер: базовый набор приложений, который можно изменять, что позволит с использованием возможностей DevOps обеспечивать положительные результаты.

Монолитные приложения: приложения, предоставляющие различные службы, которые можно развертывать только целым набором.

Мультимодальная ИТ: ИТ-среда, в которой используются некоторые модели доставки в рамках Agile и Waterfall.

Непрерывная интеграция: практика в разработке, для соблюдения которой требуется, чтобы разработчики интегрировали код в общий репозиторий несколько раз в день. Каждый раз проводится проверка во время автоматической сборки, что позволяет командам заранее выявлять неполадки.

Непрерывная доставка: ИТ-практика, ставшая популярной благодаря книге с похожим названием, где ПО автоматически проверяется на качество и развертывается далее в окружениях вплоть до среды эксплуатации.

Облако: практика использования сети серверов удаленного доступа в интернете для хранения, управления и обработки данных, приходящая на смену использованию локальных серверов.

Облакоориентированное приложение: приложение, созданное специально под стандарты и возможности облачных вычислений, в результате чего оно более стабильно и эффективно.

Образ контейнера: представление приложения в контейнере, которое можно развернуть в ПО для работы с контейнерами, чтобы иметь возможность быстро создавать приложения.

Ознакомление: начальная фаза Agile-проекта, во время которой приводятся к общему знаменателю ожидания по проекту всех заинтересованных сторон, а также способы, которые команда будет применять для достижения заданной цели.

Очки функциональности: техника оценки, регламентирующая объективный способ оценки объема работы в ИТ-проектах.

Панель наблюдения: визуальное отображение нескольких контрольных точек или отчетов, составленных на основе нескольких источников данных, для более упрощенного восприятия информации.

Паттерн заслонки: техника программирования, при которой создается новый функционал, и рабочие процессы постепенно передаются на него до тех пор, пока не появится возможность полностью отказаться от старой программы.

Плотность дефектов: метрика, измеряющая количество дефектов, обнаруживаемых в день, в программе или строках кода.

Пользовательские истории: термин, использующийся в Agile для описания функционала системы, обычно в формате «Будучи <ролью>, я хочу <функциональность>, чтобы получить <результат>».

Постепенное отключение функций: практика, согласно которой системы оказываются способными предоставлять базовую функциональность при отключении ее основных процессов; более щадящий режим, нежели полная недоступность при выходе из строя какой-либо функции.

Постфактум без обвинений: техника проведения анализа, сосредоточенная на исправлении систематических проблем без предъявления обвинений конкретным лицам.

Прикладной программный интерфейс: набор определенных методов коммуникации между различными программными компонентами, который позволяет иметь доступ к их функционалу из внешних систем.

Приложение как услуга (SaaS): программное обеспечение, предоставленное в виде сервиса в облаке с применением модели разового потребления.

Программный инкремент: период планирования, занимающий несколько спринтов/итераций. Обычно состоит примерно из пяти спринтов и длится около трех месяцев.

Продолжительность цикла: общее время от начала производственного процесса до его конца, оговоренное между вами и вашим клиентом.

Проект «с чистого листа»: проект, который команда может начать с нуля, не разбираясь с уже существующими приложениями.

Расходные сервисы: ИТ-сервисы с доступными и легко используемыми интерфейсами, которые могут быть вызваны другими программами.

Расширения IDE: расширения для интегрированной среды разработки, которые обеспечивают языки программирования полезными утилитами.

Регрессивное тестирование: комплекс тестов для подтверждения работоспособности функционала, который был добавлен ранее.

Режим «черного ящика»: тип ИТ-поставки, при котором для клиента не имеет значения, каким образом осуществляется доставка продукта, а важны лишь результаты.

Рефакторинг: практика в программировании, позволяющая программистам улучшать структуру программы без внесения функциональных изменений.

Роботизированный процесс автоматизации (RPA): техника, при которой предоставляются утилиты для автоматизации в приложениях задач, обычно выполняемых человеком вручную.

Связующее ПО (middleware): программное обеспечение, которое работает в качестве связующего между операционной системой и приложениями (например, службы интеграции и слои доступа данных).

Сдвиги конфигурации: ситуация, в которой конфигурации начинают несогласованно меняться, уходя от первоначальной конфигурации по причине вмешательства системы или человека.

Системное мышление: подход к восприятию систем в виде единого целого, а не суммы ее частей.

Системный интегратор (SI): компания, помогающая организациям собрать различные компоненты системы воедино, осуществляя реализацию,

планирование, координирование, тестирование, улучшение и иногда поддержку процессов системы.

Системы взаимодействия: быстро развивающиеся системы, с которыми пользователи взаимодействуют напрямую (см. также **Системы фиксации данных**).

Системы фиксации данных: системы, содержащие основные данные и не нуждающиеся в быстром развитии.

Составление потоков ценностей: деятельность, направленная на составление плана для трансформации организации, который будет включать в себя такие подробности, как сроки, инструментарий и др.

Среднедневная ставка: средняя стоимость рабочего дня команды с учетом нескольких значений дневных ставок на человека.

Стадия закалки: фаза Agile-проекта прямо перед развертыванием в продуктивную среду, в которой проводятся дополнительные тесты (например, производительности и безопасности), которые нельзя было провести во время выполнения спринтов.

Стек технологий: совокупность технологий, программ и компонентов, необходимых для поддержания функционала бизнеса, начиная от операционной системы вплоть до используемых приложений.

Стори-поинты: оценка работы в Agile, применяющая относительные единицы оценки вместо абсолютных, вроде дней или часов.

Сценарий командной оболочки: популярная техника автоматизации, основанная на оболочке UNIX.

Теория ограничений: научный подход к анализу систем, основанный на ограничениях, существующих в данной системе.

Теория очередей: научный подход к пониманию того, как работают очереди.

Технический долг: известные и неизвестные части программ, являющиеся на данный момент неудовлетворительными и требующими рефакторинга.

Управление конфигурацией ПО: практика отслеживания и контроля изменений в ПО, включающая в себя контроль версии, разветвление параллельной разработки и наблюдение за версией кода, включенной в программный пакет.

Уровень доступа: как правило, пользовательский интерфейс, который позволяет легче и проще получать доступ к нижележащим системам, чем при помощи систем с прямым доступом.

Уровни абстракции: разделяют два уровня архитектуры, позволяя им развиваться независимо друг от друга, не быть связанными и не вызывать появления зависимостей.

Устаревшее ПО: ИТ-термин, описывающий приложения, которые были собраны в прошлом и требуют обслуживания.

Фронтенд-команда: команда, разрабатывающая фронтенд-составляющую, которая будет напрямую обращена к клиентам.

СПИСОК ЛИТЕРАТУРЫ И ВИДЕОРЕСУРСОВ

Предисловие

1. *Mirco Hering*. Agile Reporting at the Enterprise Level (Part 2) – Measuring Productivity // Not a Factory Anymore (блог). 26 Feb 2015. notafactoryanymore.com/2015/02/26/agile-reporting-at-the-enterprise-level-part-2-measuring-productivity.

Введение

1. *Stefan Thomke and Donald Reinertsen*. Six Myths of Product Development // Harvard Business Review. May 2012. hbr.org/2012/05/six-myths-of-product-development.

2. *Don Reinertsen*. Thriving in a Stochastic World: речь на конференции YOW! 7 декабря 2015 года, Брисбен, Австралия, YouTube-видео, 56:49, опубликовано YOW! Conferences, 25 декабря 2015 года. www.youtube.com/watch?v=wyZNxB172VI.

3. The Lean Startup Methodology // The Lean Startup (сайт), запущен 10 ноября 2017 года. theleanstartup.com/principles.

4. *Brad Power*. How GE Applies Lean Startup Practices // Harvard Business Review. 23 Apr 2014. hbr.org/2014/04/how-ge-applies-lean-startup-practices.

5. *Mirco Hering*. Agile Reporting at the Enterprise Level (Part 2) – Measuring Productivity // Not a Factory Anymore (блог). 9 Nov 2015. notafactoryanymore.com/2015/11/09/lets-burn-the-software-factory-to-the-ground-and-from-their-ashes-software-studios-shall-rise.

6. *Mark Rendell*. Breaking the 2 Pizza Paradox with Platform Applications: речь на саммите DevOps Enterprise Summit 2015, Сан-Франциско, Калифорния, YouTube-видео, 25:26, опубликовано DevOps Enterprise Summit, 10 ноября 2015 года. www.youtube.com/watch?v=8WRRi6oui34.

Глава 1

1. The DevOps Platform: Overview // ADOP (DevOps-платформа Accenture на GitHub), Accenture (дата посещения 2 мая 2017 года). accenture.github.io/adop-docker-compose.

2. *Carreth Read*. Logic: Deductive and Inductive. London: DeLaMare Press, 1909. P. 320.

Глава 2

1. Gartner IT Glossary: Bimodal // Gartner, Inc. (дата посещения 2 мая 2017 года). http://www.gartner.com/it-glossary/bimodal.
2. *Ted Schadler*. A Billion Smartphones Require New Systems of Engagement // Forrester Research, Inc. (блог). 14 Feb 2012. blogs.forrester.com/ted_schadler/12-02-14-a_billion_smartphones_require_new_systems_of_engagement.
3. *Martin Fowler* Strangler Application // MartinFowler.com (блог). 29 Jun 2004. www.martinfowler.com/bliki/StranglerApplication.html.

Глава 3

1. *Mirco Hering*. How to Deal with COTS Products in a DevOps World // InfoQ (блог). 24 Jul 2016. www.infoq.com/articles/cots-in-devops-world.

Глава 4

1. *Francis Keany*. Census Outage Could Have Been Prevented by Turning Router On and Off Again: IBM // ABC News. 25 Oct 2016. www.abc.net.au/news/2016-10-25/turning-router-off-and-on-could-have-prevented-census-outage/7963916.
2. *Mike Masnick*. Contractors Who Built Healthcare.gov Website Blame Each Other for All the Problems // Techdirt (блог). 24 Oct 2013. www.techdirt.com/articles/20131023/18053424992/contractors-who-built-healthcaregov-website-blame-each-other-all-problems.shtml.

Часть Б (введение)

1. *Barry Schwartz*. The Way We Think about Work Is Broken: TED-видео, 7:42, снято в марте 2014 года в Ванкувере. www.ted.com/talks/barry_schwartz_the_way_we_think_about_work_is_broken.
2. *Dan Pink*. The Puzzle of Motivation: TED-видео, 18:36, снято в июле 2009 года в Оксфорде, Англия. www.ted.com/talks/dan_pink_on_motivation.

Глава 5

1. PI Planning // SAFe (Scaled Agile Framework). Scaled Agile, Inc. (дата обновления 11 ноября 2017 года). www.scaledagileframework.com/pi-planning.
2. *Paul Ellarby*. Using Big Room Planning to Help Plan a Project with Many Teams // TechWell Insights (блог). 26 Nov 2014. www.techwell.com/techwell-insights/2014/11/using-big-room-planning-help-plan-project-many-teams.

3. Wikipedia, Dunning–Kruger effect (дата обновления 11 ноября 2017 года, 19:01). en.wikipedia.org/wiki/Dunning%E2%80%93Kruger_effect.
4. Wikipedia, Technology tree (дата обновления 13 ноября 2017 года, 21:45). en.wikipedia.org/wiki/Technology_tree.

Глава 6

1. *Jargon File* (version 4.4.7), s.v. Conway's Law (дата посещения 14 ноября 2017 года). catb.org/~esr/jargon/html/C/Conways-Law.html.
2. 2016 State of DevOps Report. Portland: Puppet Labs, 2016). P. 9. puppet.com/resources/white-paper/2016-state-of-devops-report.
3. *Rouan Wilsenach*. DevOpsCulture // MartinFowler.com (блог). 9 Jul 2015. martinfowler.com/bliki/DevOpsCulture.html.
4. *Matthew Skelton*. What Team Structure Is Right for DevOps to Flourish? Manuel Pais (ed.) // DevOps Topologies (блог) (дата посещения 2 мая 2017 года). web.devopstopologies.com.
5. WSJF – Weighted Shortest Job First // Black Swan Farming (дата посещения 2 мая 2017 года). blackswanfarming.com/wsjf-weighted-shortest-job-first.

Глава 7

1. *W. Edwards Deming*. Out of the Crisis. Cambridge, Massachusetts: MIT Press, 1982. P. 29.
2. *Kin Lane*. The Secret to Amazon's Success Internal APIs // API Evangelist (блог, сфокусированный на API). 12 Jan 2012. apievangelist.com/2012/01/12/the-secret-to-amazons-success-internal-apis.
3. *Jeff Galimore et al.* Tactics for Implementing Test Automation for Legacy Code. Portland: IT Revolution, 2015.

Глава 8

1. Аноним, приватная беседа с автором, 2004.
2. *Dan Pink*. The Puzzle of Motivation: TED-видео, 18:36, снято в июле 2009 года в Оксфорде, Англия. www.ted.com/talks/dan_pink_on_motivation.
3. *Mark Horstman*. Managerial Economics 101: YouTube-видео, 4:33, опубликовано Manager Tools 3 мая 2009 года. www.youtube.com/watch?v=gP-RC5ZqiBg.
4. *John Goulah*. Making It Virtually Easy to Deploy on Day One // Code as Craft (блог). 13 Mar 2012. codeascraft.com/2012/03/13/making-it-virtually-easy-to-deploy-on-day-one.
5. *Mirco Hering*. Dominica DeGrandis and Nicole Forsgren, Measure Efficiency, Effectiveness, and Culture to Optimize DevOps Transformation. Portland: IT Revolution, 2015. P. 14. itrevolution.com/book/measure-efficiency-effectiveness-culture-optimize-devops-transformations.

Глава 9

1. *Jez Humble and David Farley*. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Crawfordsville, IN: Pearson Education, Inc., 2011.
2. The Netflix Tech Blog (дата посещения 16 ноября 2017 года). techblog.netflix.com.
3. Hygieia: An OSS Project Sponsored by Capital One // Capital One DevExchange (дата посещения 16 ноября 2017 года). developer.capitalone.com/open-source-projects/hygieia.

Глава 10

1. *Jez Humble*. Architecting for Continuous Delivery: речь на саммите DevOps Enterprise 2015, Сан-Франциско, YouTube-видео, 34:17, опубликовано DevOps Enterprise Summit, 17 ноября 2015 года. www.youtube.com/watch?v=_wnd-eyPoMo.
2. *Randy Shoup*. Pragmatic Microservices: Whether, When, and How to Migrate: речь на конференции YOW!, декабрь 2015 года, Брисбен, Австралия, YouTube-видео, 49:00, опубликовано YOW! Conferences, 30 декабря 2015 года. www.youtube.com/watch?v=hAwpVXiLH9M.
3. *James Lewis*. Microservices – Building Software That Is #Neverdone: речь на конференции YOW!, декабрь 2015 года, Брисбен, Австралия, YouTube-видео, 45:55, опубликовано YOW! Conferences, 29 декабря 2015 года. www.youtube.com/watch?v=JEtxmsJzrnw.
4. Wikipedia, Conway's law (дата обновления 3 ноября 2017 года, 09:02). en.wikipedia.org/wiki/Conway%27s_law.

Глава 11

1. About IT4IT // The Open Group (дата посещения 4 августа 2017 года). www.opengroup.org/IT4IT/overview.

Глава 12

1. *Keith Collins*. How One Programmer Broke the Internet by Deleting a Tiny Piece of Code // Quartz Media. 27 Mar 2016. qz.com/646467/how-one-programmer-broke-the-internet-by-deleting-a-tiny-piece-of-code.
2. *Josh Corman and John Willis*. Immutable Awesomeness: речь на саммите DevOps Enterprise 2015, Сан-Францско, YouTube-видео, 34:25, опубликовано Sonatype 21 октября 2015 года. www.youtube.com/watch?v=-S8-lrm3iV4.
3. *Debbi Schipp*. Bonus Bet Offers Peak as Online Agencies Chase Cup Day Dollars // News.com.au. 1 Nov 2016. www.news.com.au/sport/superracing/melbourne-

cup/bonus-bet-offers-peak-as-online-agencies-chase-cup-day-dollars/news-story/8e09a39396fb5485cf1f24cbea228ff9.

4. *Yury Izrailevsky and Ariel Tseitlin*. The Netflix Simian Army // The Netflix Tech Blog.18 Jul 2011. techblog.netflix.com/2011/07/netflix-simian-army.html.

Приложение

1. *Mirco Hering*. Agile Reporting at the Enterprise Level (Part 2) – Measuring Productivity // Not a Factory Anymore (блог). 26 Feb 2015. notafactoryanymore.com/2015/02/26/agile-reporting-at-the-enterprise-level-part-2-measuring-productivity.

2. *Andy Boynton and William Bole*. Are You an 'I' or a 'T'? // Forbes Leadership (блог). 18 Oct 2011. www.forbes.com/sites/andyboynton/2011/10/18/are-you-an-i-or-a-t/#2517d45b351b.

3. *Don Reinertsen*. Thriving in a Stochastic World: речь на конференции YOW!, 7 декабря 2015 года, Брисбен, Австралия, YouTube-видео, 56:50, опубликовано YOW! Conferences, 25 декабря 2015 года. www.youtube.com/watch?v=wyZNxB172VI.

4. *Gary Gruver and Tommy Mouser*. Leading the Transformation: Applying Agile and DevOps Principles at Scale. Portland: IT Revolution, 2015. P. 17.

5. *Frederick P. Brooks, Jr.* The Mythical Man-Month: Essays on Software Engineering, anniversary ed., 2nd ed. Crawfordsville: Addison-Wesley Longman, Inc., 2010). P. 25.

6. *Frederick P. Brooks, Jr.* The Mythical Man-Month. Addison-Wesley Longman, Inc., 1995. P. 17.

Книги издательства «ДМК Пресс»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК Пресс», «СОЛОН ПРЕСС», «КТК Галактика»).
Адрес: г. Москва, пр. Андропова, 38;
тел.: **(499) 782-38-89**, электронная почта: **books@alians-kniga.ru**.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.

Мирко Херинг

DevOps для современного предприятия

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Райтман М. А.*
Редактор *Готлиб О. В.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 18,85. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**