ГЛАВА 2
ЗАКОН МИКРОКОМАНДЫ

Фундаментальная цель — достичь «мелкости» в крупных организациях.

Эрнст Фридрих Шумахер[1]

Представьте себе следующую ситуацию. На дворе 1997 год, вам только что пришла в голову отличная идея — создать тонкое портативное устройство, которое помещается в карман и выполняет множество функций посредством прикосновения пальца. Это одновременно портативный телефон, адресная книга, карта, навигатор, посадочный талон, музыкальный и видеопроигрыватель, телевизор, камера, фонарик, диктофон, секундомер, будильник, переводчик, пульт дистанционного управления, хранилище газет и журналов со всего мира, библиотека с тысячами книг и так далее. Более того, устройство молниеносно выполняет любые запросы, и его можно адаптировать под индивидуальные потребности и предпочтения сотен миллионов человек со всего мира.

Звучит неплохо, не так ли? Вы уверены в беспроигрышности идеи. Но как ее воплотить? Что ж, сейчас 1997 год, и вы применяете лучшие на этот период управленческие практики. Первые несколько лет уходят на то, чтобы заверить всех в осуществимости идеи и убедить

стратегический комитет крупной корпорации в разработке и выведении на рынок вашего нового потрясающего устройства. Когда одобрение наконец получено, вы нанимаете огромную команду дизайнеров и разработчиков. Последующие несколько лет команда занимается разработкой детализированных спецификаций устройства и строит график поставки и интеграции его различных компонентов. Затем вы берете на работу сотни тысяч инженеров и прочих специалистов, чтобы создать это устройство, и десятки тысяч менеджеров, которые будут управлять и контролировать действия технических сотрудников — в соответствии с четким планом. Вы вводите масштабные системы отчетности, чтобы высшее руководство знало, на что потрачен каждый доллар и насколько продвинулась работа над каждым компонентом. Вы создаете систему комитетов по координации, чтобы убедиться, что все многочисленные подразделения, словно слаженно играющий симфонический оркестр, трудятся над созданием устройства.

Что в итоге? Спустя многие годы, потратив миллиарды долларов, вы обнаруживаете, что ваше устройство по-прежнему не готово к выходу на рынок. Технические проблемы не устраняются, а, похоже, лишь растут в геометрической прогрессии. Координация работы между подразделениями вызывает огромную головную боль, несмотря на все усилия, приложенные соответствующими комитетами. Идет ожесточенный поиск виноватых в технических проблемах и задержках. Отдельные компоненты кажутся многообещающими, но зачастую несовместимы между собой. Подразделения обвиняют друг друга, но отрицают собственную вину. Устранение проблем занимает вечность: как только найдено решение одной, возникает другая.

Организовав работу подобным образом, вы надеялись достичь эффекта масштаба. Но результаты в корне противоположны. Затраты, связанные с организацией и координированием работы такого большого числа сотрудников, растут гораздо быстрее, чем любая добавочная стоимость, ими же создаваемая. Окончание работы по-прежнему остается под вопросом[2].

К сожалению, несмотря на затраченные годы, усилия и финансы, руководство принимает решение прикрыть ваш проект. Дело в том, что конкурент — Apple — к 2007 году разработал схожий продукт «с нуля до выхода на рынок». На это компании понадобилось всего 18 месяцев и гораздо меньшие затраты на традиционный менеджмент.

Как ей это удалось? Вместо того чтобы создавать очень сложное устройство с очень сложной организацией согласно разработанным спецификациям, Apple поступила с точностью до наоборот. Она создала сравнительно простое физическое устройство — iPhone. При этом она работала короткими циклами, постоянно совершенствуя продукт. Компания не стала нанимать сотни тысяч инженеров, чтобы создать монолитное ПО для выполнения функций устройства, а подготовила технологическую платформу и пригласила сотни тысяч независимых команд, которые могли создавать собственные специализированные приложения, удовлетворяющие любые потенциальные потребности пользователя. Команды разработчиков предлагают их потребителям Apple напрямую. Все эти приложения выполняют практически бесконечное множество функций. Они добавляются по мере разработки, при этом их авторы напрямую взаимодействуют с пользователями. Последние решают, какие приложения отвечают их потребностям. В результате появилось многофункциональное устройство, которое адаптируется под индивидуальные предпочтения и сиюминутные прихоти сотен миллионов отдельных пользователей. Совершить такой подвиг традиционными методами управления невозможно. Вместо раздувания организации для решения сложной задачи Apple минимизировала проблему, разбив ее на крошечные части, с которыми вполне могут справиться маленькие независимые команды, работая короткими циклами и постоянно получая обратную связь от пользователей.

Успех iPhone компании Apple часто приписывают гению Стива Джобса. Блестящему маркетингу. Превосходному дизайну. Тщательному вниманию к деталям. Инновационному мышлению. Огромному стремлению

решать проблемы. Все это верно. Однако зачастую из виду упускается тот факт, что все эти элементы были бы бесполезны, если бы Apple методом итераций не разработала сравнительно простое электронное устройство и не создала бы платформу для независимых разработчиков ПО. Сотни тысяч микрокоманд проявили свои мастерство и талант, чтобы создавать приложения, напрямую взаимодействуя с потребителями.

***

Закон микрокоманды прост. Он предполагает, что для решения масштабных и сложных задач в VUCA-мире компаниям следует не создавать раздутые структуры, а по мере возможности разбивать эти задачи на мелкие блоки и передавать их кросс-функциональным автономным командам, которые работают методом итераций, то есть короткими циклами в состоянии потока. При этом они получают быструю обратную связь от потребителей и конечных пользователей.

История о провальных попытках создать портативное устройство при помощи традиционных управленческих практик, описанная в начале главы, выдумана. Однако некоторые ее детали подозрительно напоминают мытарства Newton, личного цифрового ассистента, за создание которого в 1987 году отвечал Джон Скалли, бывший на тот момент CEO Apple. В 1998 году Стив Джобс прекратил разработки Скалли[3].

То, что попытки решать сложные проблемы в рамках нисходящей бюрократии часто заканчиваются крахом, — это факт. Например, в 2006 году ВВС США запустили проект по модернизации менеджмента логистики. Они заключили контракт на сумму $628 млн с Computer Sciences Corporation, которая выполняла в проекте роль системного интегратора. Компания занималась конфигурацией, разработкой, проведением обучения и управлением преобразованиями перед запуском[4].

«Мы никогда не пытались одновременно изменить процессы, инструменты и язык всех 250 тысяч сотрудников нашего бизнеса, — говорил Гровер Данн, директор отдела по преобразованиям ВВС. — Но сейчас

мы собираемся сделать именно это»[5]. Элизабет Макграт, заместитель начальника по вопросам управления военного ведомства, поясняла: «Мы начали с одномоментной реорганизации и определили все возможные требования к программе, что сделало ее громоздкой и сложной»[6].

На протяжении последующих семи лет проект несколько раз реструктурировался. В 2013 году BBC подсчитали: на воплощение одной четверти изначально запланированных возможностей потребуется еще $1 млрд, и даже при этих огромных вложениях на запуск системы уйдет еще семь лет. В итоге проект, затраты на который уже составили $1,3 млрд, закрыли.

***

Возможно, вы скажете, что разделение работы на микрочасти имеет смысл, лишь когда речь идет о персональных устройствах вроде iPhone. Правомерен ли такой подход в отношении серьезного промышленного проекта, например для создания истребителя нового поколения? Да, правомерен, и тому уже есть подтверждение. Так, шведский производитель воздушных судов Saab разработал истребитель Gripen именно на основе Agile-практик[7]. Каждые полгода Saab объявляет о новом выпуске операционной системы истребителя. Она делает его быстрее, дешевле, легче, эффективнее, мощнее, улучшает электронику и обеспечивает более продвинутые системы наведения. Ведущие эксперты по вопросам обороны назвали Gripen «лучшим малозаметным истребителем в мире»[8].

Авторитетная международная издательская группа IHS Jane's, специализирующаяся на военной тематике, провела исследование, сравнив операционные издержки детища Saab с издержками истребителей F-16 и F-35 (Lockheed Martin), F/A-18 Super Hornet (Boeing), Rafale (Dassault) и Typhoon (Eurofighter). И пришла к выводу, что Gripen имел «наименьшие операционные расходы среди всех изученных истребителей». При этом рассматривались такие факторы, как расход топлива, предполетная подготовка и ремонт, а также регулярное аэродромное техобслуживание совместно с сопутствующими затратами на персонал»[9].

Программное обеспечение играет все более значимую роль в разработке и эволюции Gripen. «Головная боль, с которой сталкиваются производители истребителей, — пишет Билл Свитман в Aviation Week and Space Technology, — заключается в том, что каким бы грамотным ни был ваш проект, разработка и создание этих самолетов ведет к огромным расходам. Кроме того, их жизненный цикл от проектирования до утилизации намного превосходит рамки политического или технологического горизонта». Gripen был разработан с учетом этих особенностей: «Продолжительный срок службы требует гибкости как в отношении боевых миссий, так и на протяжении жизненного цикла»[10].

Тот же феномен наблюдается в мире автомобилей. На протяжении первого века существования автомобилей человек покупал машину без всякой возможности модифицировать впоследствии свое приобретение, например увеличив его мощность или повысив функциональность. Теперь ситуация изменилась. В частности, Tesla может дополнить новыми функциями — автоматическим торможением при вероятном столкновении, частичной системой автопилота и автоматизированной системой парковки — уже проданные автомобили, загрузив в них новое ПО. Подобные возможности предлагаются и другими производителями автомобилей класса люкс вроде Audi или Mercedes-Benz. Разница лишь в том, что Tesla Model S разработана с возможностью постоянного апгрейда «на лету».

«Model S, по сути, представляет собой очень сложный компьютер на колесах, — рассказывает CEO компании Илон Маск. — Tesla разрабатывает как ПО, так и аппаратное обеспечение. Мы относимся к апгрейду автомобиля так же, как к обновлению телефона или ноутбука»[11].

Четырехколесные «железные кони» все больше напоминают гибкие электронные устройства, а не механические сооружения. Как и iPhone, автомобиль становится платформой для приложений.

В этой связи полезно вспомнить, что Agile-менеджмент начал ассоциироваться с программным обеспечением лишь с 2001 года. Исторические же

его корни лежат в сфере производства — в quality movement в Японии, в частности в Toyota Production System. Компания начала с того, что стала экспериментировать с маленькими промышленными сериями и обнаружила, что работа короткими циклами, определяемыми спросом, оказалась более эффективной, чем массовое производство[12].

Такая модель распространилась в Японии в 1970-х годах и в 1980-х добралась до США. Обнаружилось, что при правильном ее применении длительность производственного цикла с учетом мелких партий можно сократить в 10–100 раз, а объем запасов — более чем на 90%, что высвобождало огромные суммы денег. К «вторичным» эффектам относились повышение качества, ускорение обучения и снижение себестоимости продукта.

Итерационный подход, распространенный затем на другие производственные сферы, описан в статье Хиротаки Такеучи и Икуджиро Нонаки The New New Product Development Game, опубликованной в Harvard Business Review в 1986 году. Авторы писали:

> Компании все больше осознают, что старый последовательный метод разработки новых продуктов попросту не работает. Вместо него японские и американские компании используют *целостный* подход. Как в регби, мяч передается между членами команды, которые слаженно передвигаются по полю.
>
> Этот целостный подход обладает шестью характеристиками: встроенной нестабильностью, самоорганизующимися проектными командами, параллельными фазами разработки, мультиобучением, неявным контролем и передачей знаний внутри организации. Шесть элементов складываются, словно пазл, в быстрый, гибкий процесс разработки нового продукта. Не менее важен тот факт, что новый подход может инициировать перемены. Это инструмент создания креативных идей и процессов, ориентированных на рынок, внутри существующей организации[13].

В качестве примеров в статье приводятся Fuji-Xerox, Honda и Canon, разрабатывающие аппаратное, а отнюдь не программное обеспечение.

В 1990 году итерационный микрокомандный подход распространился шире. В классической книге «Машина, которая изменила мир»* он получил название «бережливое производство»[14]. Однако, зародившись в сфере аппаратного обеспечения, систематическое использование микрокоманд и итерационных подходов получило подлинное признание именно в сфере разработки ПО после публикации Agile-манифеста в 2001 году.

***

Какие именно практики входят в Закон микрокоманды? Зависит от того, с какой стороны посмотреть. В течение первого десятилетия после публикации Agile-манифеста его приверженцы яростно спорили друг с другом, пытаясь определить «настоящие Agile-практики». Одни называли Scrum. Другие твердо верили, что это Канбан, третьи — что это Lean (бережливое производство). В конце концов стало понятно, что правильный ответ звучит иначе. Закон микрокоманды подразумевает образ мышления, а не конкретный набор инструментов и процессов, которые можно перечислить в практическом руководстве. Если вы относитесь к Agile как к последнему, вы не стоите на правильном пути. Нельзя прийти в магазин и «купить немного Agile-менеджмента».

Закон микрокоманды определяет, как должна выполняться сложная работа. Практики возникают как результат сочетания Agile-мышления с особыми организационными условиями. Именно по этой причине сотрудничество с консалтинговой фирмой, сотрудники которой «придут и научат наш персонал инструментам и процессам Agile-управления», редко приводит к успеху.

---

* Джеймс П. Вумек, Дэниел Т. Джонс, Дэниел Рус. Машина, которая изменила мир. М.: Попурри, 2007. *Прим. ред.*

издательство
МАНН, ИВАНОВ И ФЕРБЕР
Максимально полезные книги