

Джо Мараско



IT ПРОЕКТЫ ФРОНТОВЫЕ ОЧЕРКИ

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-096-0, название «ИТ-проекты: фронтовые очерки» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

The Software Development Edge

Essays on Managing Successful Projects

Joe Marasco

◆◆ Addison-Wesley

ПРОФЕ  ИОНАЛЬНО

IT-проекты

фронтальные очерки

Эссе об управлении успешными проектами

Джо Мараско



Санкт-Петербург — Москва
2008

Серия «Профессионально»
Джо Мараско
IT-проекты: фронтовые очерки

Перевод С. Маккавеева

Главный редактор	<i>А. Галунов</i>
Зав. редакцией	<i>Н. Макарова</i>
Выпускающий редактор	<i>В. Овчинников</i>
Научный редактор	<i>О. Цилюрик</i>
Редактор	<i>В. Овчинников</i>
Корректор	<i>О. Макарова</i>
Верстка	<i>Д. Орлова</i>

Мараско Дж.

IT-проекты: фронтовые очерки. – Пер. с англ. – СПб: Символ-Плюс, 2007. – 384 с., ил.

ISBN-10: 5-93286-096-0

ISBN-13: 978-5-93286-096-0

Этот сборник, написанный живо и интересно, содержит практические рекомендации по управлению проектами, образованные в результате синтеза принципов управления и богатейшего практического опыта автора, и призван способствовать совершенствованию управления проектами в любой softверной (и не только) компании. Затронуты разнообразные аспекты создания ПО и управления этим процессом, высказаны оригинальные идеи о сущности программного обеспечения, о том, как мотивировать профессионалов, и о многом другом. Эта книга будет полезна не только тем, кто занят в производстве ПО. Ее с интересом прочтет менеджер любого уровня.

ISBN-10: 5-93286-096-0

ISBN-13: 978-5-93286-096-0

ISBN 0-321-32131-6 (англ)

© Издательство Символ-Плюс, 2007

Authorized translation of the English edition © 2005 Pearson Education, Inc. This translation is published and sold by permission of Pearson Education, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7, тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Налоговая льгота – общероссийский классификатор продукции ОК 005-93, том 2; 953000 – книги и брошюры.

Подписано в печать 23.11.2007. Формат 70х90¹/₁₆. Печать офсетная.

Объем 24 печ. л. Тираж 2000 экз. Заказ N

Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»

199034, Санкт-Петербург, 9 линия, 12.

Посвящается Уини, свету моей жизни.



Оглавление

	Похвальные отзывы на «The Software Development Edge».....	14
	Об авторе	18
	Благодарности	19
	Предисловие.....	23
	Введение	26
Часть I	Общий менеджмент	29
<i>Глава 1</i>	<i>В начале начал</i>	<i>31</i>
	Значение качества программ	32
	Островки надежности.....	33
	Для кого предназначена эта книга	35
	Циферблат для итеративного решения задач.....	36
	Резюме.....	39
<i>Глава 2</i>	<i>Мои вычислительные корни.....</i>	<i>40</i>
	Ниспровергатель	41
	Ответ.....	42
	Как действовала эта программа	43
	Что было особенного в этом поколении инженеров?.....	45
	Вычисления	47
	Тесное знакомство с числами.....	48
	Так что там с этими компьютерами?.....	50
	Наше вычислительное наследство	51
	Резюме.....	52
<i>Глава 3</i>	<i>Альпинизм</i>	<i>55</i>
	О восхождении на высокие горы.....	56

	Обычные причины провалов	64
	Составляющие успеха	65
	Человеческий фактор.....	67
	Резюме.....	67
<i>Глава 4</i>	<i>Менеджмент</i>	68
	Управление командами.....	69
	Резюме.....	77
Часть II	Особенности разработки ПО.....	79
<i>Глава 5</i>	<i>Самое важное</i>	81
	Итеративная разработка	81
	Роско Леруа	82
	Оставляем последовательный подход.....	84
	Другая крайность	85
	Первый рисунок Роско	86
	Второй рисунок Роско.....	87
	Минуточку!.....	88
	Укорачиваем векторы.....	89
	Применение к разработке ПО	89
	Обучение на практике и выбор коротких векторов	90
	Программирование рисков	91
	А эту песню ты слышал?	92
	Еще об обучении на практике.....	94
	Следствия для бизнеса	95
	Эффект численности персонала	96
	Простой здравый смысл.....	98
	Резюме.....	99
<i>Глава 6</i>	<i>Моделирование</i>	101
	Как рассказывать об UML.....	103
	Что такое UML и в чем его значение?	103
	Второй, менее тривиальный пример	104
	Третий пример.....	106
	А теперь вспомним о программах.....	108
	Выходим а новый уровень абстракции.....	109
	Резюме.....	109
<i>Глава 7</i>	<i>Написание кода</i>	111
	Как менеджеру изучить новый язык программирования ...	112
	Задача, более точная формулировка.....	113

	Что должно быть в стандартной задаче?	114
	Игра в животных	115
	Удовлетворяет ли игра в животных критериям?	116
	Языки, прошедшие тест	118
	Это ваша игра	119
	Резюме	119
<i>Глава 8</i>	<i>За дверь его!</i>	121
	Если Вы его постройте, они придут	123
	Вначале была песочница	123
	Почему же сборка продукта так трудна?	124
	Как насчет итеративной разработки?	131
	Резюме	132
Часть III	Разработка ПО с точки зрения	
	управления проектами	133
<i>Глава 9</i>	<i>Компромиссы</i>	135
	Пирамида проекта	136
	Пять, а не четыре	138
	Появляется пирамида	139
	Переменная высота	140
	Объем пирамиды представляет собой константу	140
	Статистическая интермедия	141
	Идея правильная, распределение – нет	144
	Приложение к реальным проектам	146
	Что надо, чтобы сыграть в «орлянку»?	147
	Рост уверенности	147
	Важные предостережения	149
	Все дело в риске	154
	Резюме	155
<i>Глава 10</i>	<i>Оценка времени</i>	157
	Не прибегнуть ли к здравому смыслу?	158
	Шоколадное или ванильное?	158
	Разъяснения Роско	159
	Роско идет дальше	160
	Календарь Роско	160
	Роско вычисляет	161
	Роско обращается к программированию	161
	Роско докладывает	162

	Кое-что мы делаем правильно	162
	Роско подытоживает	163
	Роско расправляется со всеми	164
	Кое-что мы делаем правильно, часть вторая	165
	Роско допущен в компанию менеджеров программных проектов	166
	Резюме	166
<i>Глава 11</i>	<i>График работ</i>	167
	Роско формулирует задачу: насколько вы опоздаете?	168
	Джо отчасти возвращает свои позиции	169
	Роско возвращается	170
	Список негодяев по Роско	171
	График Роско	172
	Последнее возражение	173
	Заключительный выстрел Роско	174
	Резюме	174
<i>Глава 12</i>	<i>Ритм</i>	176
	Взгляд физика на течение проекта	177
	Вмешательство реальности	187
	А что с итеративной разработкой?	188
	Последний график	194
	Резюме	196
Часть IV	Человеческий фактор	197
<i>Глава 13</i>	<i>Политика</i>	199
	Контекст	200
	Определение	201
	Три сценария	202
	Политика неизбежна, но.....	203
	Когда обстановка политизируется	205
	Соответствие между представлениями инженеров и обычными представлениями	208
	Среда с высокой степенью доверия	209
	Другие разновидности плохой политики	210
	Резюме	212
<i>Глава 14</i>	<i>Ведение переговоров</i>	214
	Общение – это все	214
	Роско излагает свою теорию	215

	Ну, теперь все?	223
	Резюме	224
Глава 15	Получение согласия	226
	Роско разбивает себе нос.....	227
	...И сразу переходит к делу	227
	Извержение Везувия	227
	Как это делается в Техасе	228
	Как это связано с программным обеспечением	229
	Мое домашнее задание съела собака	230
	Войны спецификаций?	231
	Три самых распространенных отговорки	232
	И еще одно.....	234
	Удар, защита, ответный удар	235
	Жертва большого проекта	236
	Конец программных разработок в прежнем понимании? ..	236
	Проработка и построение системы	237
	Жестокая любовь	238
	Резюме	238
Глава 16	Вознаграждение	240
	Ищем поток	241
	Поток и продуктивность разработки ПО	243
	Применение модели потока к вознаграждению	244
	Деньги не всегда помогают	255
	Резюме	256
Часть V	Нестандартный подход	259
Глава 17	Урок истории	261
	Не берите в архитекторы короля	262
	Внешнее впечатление обманчиво	262
	Контроль проекта	262
	Знать, чего ты не умешь	263
	Преемственность руководства	263
	Как всегда, все в спешке	263
	Сосредоточенность на несущественных деталях	263
	Если проект плох.....	264
	Важность тестирования	264
	Прототип и продукт	264
	Выводы следствия	265

	Резюме.....	265
<i>Глава 18</i>	<i>Неправильные аналогии</i>	266
	Хьюстон, у нас проблемы	266
	Ложные законы Ньютона	268
	Все относительно.....	271
	Квантовая чушь	274
	Тепловая смерть	278
	Другие примеры	281
	Хорошая наука	281
	Резюме.....	282
<i>Глава 19</i>	<i>Проблема обновления</i>	283
	Обновление встроенного ПО	285
	Современная ситуация	285
	Игры вокруг обновления ПО	286
	Скромное предложение.....	287
	Еще раз об обновлении ПО	288
	Некоторые приятные следствия.....	289
	На чем основана жизнеспособность такой схемы	290
	Уточнение	291
	Как быть с программным пиратством?	292
	Пока солнце не взяло верх.....	293
	Резюме.....	293
<i>Глава 20</i>	<i>Числа случайные и не очень.....</i>	295
	Роско описывает ситуацию.....	296
	Моделирование отбивающего	296
	Первые шаги.....	298
	Следующие шаги.....	300
	Получение дополнительных вероятностей	301
	Про бейсбол мы, конечно, уже забыли	303
	Действительность уродлива	303
	Решение Понедельника	304
	Полученные уроки	308
	Резюме.....	310
Часть VI	Более сложные вопросы	311
<i>Глава 21</i>	<i>Кризис</i>	313
	Пять дней рыбы.....	314
	Рыбный рынок	314

	День 1: Неведение	315
	День 2: Прячемся от проблемы	315
	День 3: Те же и «наладчик».....	315
	День 4: Поворотный момент	316
	День 5: Два критических маршрута	316
	Мораль	317
	Резюме.....	317
<i>Глава 22</i>	<i>Расширение</i>	<i>319</i>
	Проблемы роста.....	319
	Наивная модель	321
	Модель и действующие в ней допущения.....	323
	Наглядный пример.....	330
	Нелинейность.....	332
	Призыв к действию	334
	Выводы	335
	Номограмма.....	337
	Электронная таблица.....	339
	Резюме.....	340
<i>Глава 23</i>	<i>Культура.....</i>	<i>341</i>
	Что такое культура?	342
	Сильные и слабые культуры	343
	Определение корпоративных ценностей	344
	А в применении к ПО... ..	347
	Создание сильной культуры.....	349
	Если вы ищете работу.....	354
	Краткие итоги.....	356
	Резюме.....	356
<i>Глава 24</i>	<i>Соединяем все вместе</i>	<i>357</i>
	Шлеппер	358
	Махер.....	361
	Менш	363
	Еще о меншах	365
	Распределение людей по группам.....	366
	Некоторые завершающие мысли по поводу модели.....	367
	Резюме.....	368
	<i>Алфавитный указатель</i>	<i>371</i>



Похвальные отзывы на «The Software Development Edge»

«В этих статьях сконцентрированы десятилетия практического опыта разработки ПО в широком спектре предметных областей. Джо Мараско достаточно закален в боях, чтобы сформулировать рецепты достижения успеха, пригодные для широкой аудитории. В его арсенале математика, физика, здравый смысл, мастерство рассказчика и лишенный приторности стиль, которые помогают ему изложить свою уникальную точку зрения на важные проблемы поставки законченных программных продуктов. Любой читатель – исследователь в области вычислительных наук, отчаявшийся менеджер программного проекта, успешный бизнесмен *или скептически настроенный программист* – *найдет массу полезного в этой подборке.*»

– *Уокер Ройс (Walker Royce), автор «Software Project Management: A Unified Framework» (Addison-Wesley)*
и вице-президент IBM Software Services-Rational

«Легко читаемые очерки Джо Мараско об управлении успешными проектами демонстрируют, что менеджеры программных разработок – и в этом они не отличаются от всех других менеджеров, – чтобы преуспеть, должны овладеть основами менеджмента: работать с людьми и проявлять решительность, решать политические проблемы, придерживаться графика работ и, конечно, поставлять продукт хорошего качества. Простым языком Мараско объясняет многие сложные виды деятельности, начиная с оценки времени, реально необходимого для выполнения работы, и заканчивая успешным ведением переговоров, и даже красноречиво описывает три от-

дельные фазы нашего личного развития. Он часто облекает повествование в форму беседы с вымышленным коллегой – Роско Леруа, и такие сократовские диалоги позволяют ему иллюстрировать две разные точки зрения во многих областях (вспоминаются труды Галилея, содержащие еретические по тем временам взгляды). Рекомендации Мараско помогут профессионалам в области технологий справиться со всепроникающей властью некомпетентности и дадут читателям средства более эффективного решения задач, которые ставит перед нами наш непрерывно развивающийся мир».

– *Карл Селинджер (Carl Selinger), автор «Stuff You Don't Learn in Engineering School: Skills for Success in the Real World» (Wiley-IEEE Press) и автор-редактор журнала IEEE*

«Эта книга насыщена практическим опытом, послужившим основой глубоких мыслей, которые будут полезны каждому менеджеру программных разработок или вдумчивому инженеру-программисту. К управлению программными проектами Джо применяет свои меткие наблюдения в области инженерии, физики, программирования и менеджмента, равно как практические технологии и методы решения задач, необходимые для достижения успеха».

– *Джон Ловитт (John Lovitt), старший вице-президент Rational Software (в отставке)*

«Эта книга будет полезна не только тем, кто занимается программным бизнесом. Любой менеджер любого уровня может с интересом начать читать ее с любого места и в любое время. Настоятельно рекомендуемая для чтения книга».

– *Р. Макс Уайдман (R. Max Wideman), сотрудник Project Management Institute, AEW Services, Ванкувер, Канада*

«Книга Мараско окажется захватывающим чтением для всякого, кто интересуется общими проблемами управления. Читатель познакомится с простыми количественными моделями, основанными на опыте автора, которые помогут оценить производительность и выбрать путь к успешному завершению сложного проекта. Кроме того, и сами истории и стиль автора делают чтение как занимательным, так и информативным».

– *Мартин Лессер (Martin Lesser), профессор факультета механики, Королевский технологический институт (КТН), Стокгольм*

«Каждая глава в отдельности написана ясно, занимательно, со знанием дела, наводит на размышления и информативна. В совокупности, как сборник коротких рассказов опытного и умелого автора, эти главы предлагают читателю и учат его видеть и понимать то, на что часто бросают лишь беглый взгляд и едва замечают. Талант Мараско к анализу и синтезу, позволяющий ему увидеть общую картину в мелочах и выделить важнейшие детали в масштабной панораме, одновременно просвещает и впечатляет».

– *Стивен Д. Франклин (Stephen D. Franklin),
Калифорнийский университет, Ирвайн*

«Практические рекомендации по управлению проектами, написанные занимательно и демонстрирующие эрудицию автора. Его мысли представляют собой впечатляющий синтез принципов управления и практического опыта и будут способствовать совершенствованию управления проектами в любом виде организационной деятельности».

– *Стивен Глоberman (Steven Globberman), профессор международного
бизнеса, университет штата Западный Вашингтон*

«Джо – лидер, понимающий эмоции, и проницательный аналитик целостных систем. В книге ярко проявились оба эти качества».

– *Йоси Амрам (Yosi Amram), коуч топ-менеджеров,
основатель и CEO, Individual Inc. и Valicert Inc.*

«Джо Мараско собрал будоражащий мысль шведский стол, который придется по вкусу всякому, кто работает с программными проектами и разработчиками ПО. Откройте книгу в любом месте и вы найдете новые идеи о сущности ПО, о том, как мотивировать профессионалов, или по какой-то другой теме, которые заставят вас пересмотреть свои воззрения. Необязательно соглашаться со всем, что пишет Джо, но его позиция вызывает уважение, и какие-то из своих представлений вы, возможно, измените. Многие из глав книги войдут в список обязательной литературы для моих студентов, изучающих разработку ПО».

– *Гэри Полис (Gary Pollice), преподаватель вычислительной
техники, Вустерский политехнический институт*

«Необычный и эмоциональный философский труд, мудрость которого основана как на академическом, так и на деловом опыте... Практическое, революционное и остроумное руководство по управлению тонкими нитями и машин, и человеческих душ. Джо показывает, что самая важная составляющая успеха и счастья в бизнесе и в жизни – это честность».

– Кейт Джонс (*Kate Jones*),
президент *Kadon Enterprises, Inc.* (www.gamepuzzles.com)

«Новая книга Джо Мараско может служить не только надежным введением в управление проектами как научную дисциплину, но и занимательным чтением. С помощью простых и легких для понимания и изложения примеров ему удастся вскрыть и описать базовые проблемы формирования команд и управления людьми и проектами. Эту книгу надо обязательно прочесть тем, кто хочет глубже разобраться в данной науке».

– Борис Люблински (*Boris Lublinsky*),
Enterprise Architect, CNA Insurance

«Любой менеджер, который не зря получает свои деньги, не станет искать спасения в «практическом руководстве». Но он не откажется от пары золотых самородков, которые спасут его, если он попадет в беду, или укрепят его дух, помогут поверить, что он на правильном пути, или разбудят его мысль, призванную осветить выход из затруднения. Мараско показывает менеджеру путь к целой жипе таких самородков».

– Билл Ирвин (*Bill Irwin*),
руководитель в отставке, индустрия высоких технологий



Об авторе

Джо Мараско – ветеран, отдавший индустрии ПО 35 лет, сейчас занимает пост президента и CEO компании Ravenflow, а до этого был первым вице-президентом и руководителем Rational Software, ставшей одним из пяти брендов IBM. Он занимал многочисленные ответственные должности, связанные с разработкой продуктов, маркетингом и организацией продаж в регионах, руководя проектами Rational Apex и Visual Modeler для Microsoft Visual Studio. В 1998 г. он занимал должность первого вице-президента по операционной деятельности, а покинул Rational в 2003 г. Имеет степень бакалавра химической технологии от Cooper Union, степень Ph.D. по физике от университета Женевы (Швейцария) и M.S.A. (магистра бухгалтерского учета), полученную по окончании аспирантуры по менеджменту в университете Калифорнии в Ирвайне. В свободное от сочинительства и работы время он жарит барбекю и играет в гольф. Ребрышки в его исполнении намного приятнее, чем впечатления от счета в партиях, иггранных против него.



Благодарности

Непонятно даже, как *приступить* к этой задаче. Моя книга основана на профессиональном опыте, полученном в течение нескольких десятилетий, на протяжении которых, можете мне поверить, я встречался с очень многими людьми, оказавшими на меня влияние. Я решил представить их в хронологическом порядке.

Средняя школа в Фар Рокауэе: Амелия Векслер (Amelia Wexler), научившая меня писать.

Куперовский союз (The Cooper Union): Кларенс Шерман (Clarence Sherman)^{*}, Боб Стейнбергер (Bob Steinberger), Мэри Блэйд (Mary Blade)^{*} и Чарльз Ричард Экстерманн (Charles Richard Extermann)^{*}. Каждый из них старался научить меня быть человеком не меньше, чем химиком, инженером и физиком. С Бобом Стейнбергером, который был свидетелем на моей свадьбе, мы поддерживаем дружеские отношения уже более 40 лет.

Нью-Йоркский университет в Стони Брук (Stony Brook): Леонард Айзенбуд (Leonard Eisenbud) и Альфред Гольдхабер (Alfred Goldhaber).

Швейцария и Франция: Пьер Экстерманн (Pierre Extermann), Оливье Гизан (Olivier Guisan), Рональд Мермо (Ronald Mermoud), Мишель Пушон (Michel Pouchon), Мишель Исперьян (Michel Isperian) и Рене Тюрлэ (René Turlay)^{*}. Они помогли мне получить степень PhD за один заход, и я не знаю, какая благодарность за это была бы достаточной.

^{*} Ныне покойные.

Ирвайн: Стив Франклин (Steve Franklin), Альфред Борк (Alfred Bork), Боб Додж (Bob Dodge) и Джо ЛаРоза (Joe LaRosa). Сначала в Калифорнийском университете в Ирваине, а потом в корпорации Fluor я осуществлял переход из академического мира в мир настоящей жизни. Они были моими проводниками на этом пути. Боб Додж, например, показал мне, что значит быть хорошим начальником. Майк Кинсмен научил меня работать с деловыми показателями и остается моим добрым другом и надежным советником уже больше 25 лет.

Сан-Матео: Билл Бьюкенан (Bill Buchanan), обративший мое внимание на Кремниевую Долину.

Лаборатории Мак-Леода: Билл Ирвин (Bill Irwin), показавший мне тонкости управления начинающими компаниями и убедивший меня в том, что они составляют общую проблему управления. Дэннис Аллисон (Dennis Allison), всегда находившийся рядом, когда требовалась его помощь – даже когда я не знал, что она требовалась. Нельзя не упомянуть и Пола Хвошински (Paul Hwoshinsky), открывшего для меня понятие неденежных активов.

Rational: ее основатели Майк Дэвлин (Mike Devlin) и Пол Леви (Paul Levy), сочетание увлеченности и интеллекта которых оказывало вдохновляющее воздействие. Благодаря Майку и Полу целое поколение людей достигло большего, чем то, на что они, как им казалось, были способны.

В Rational меня учили маркетингу: Бретт Бахман (Brett Bachman), Йоси Амрам (Yosi Amram) и Джерри Рудисин (Jerry Rudisin).

В Rational меня учили работе с клиентами: Боб Бонд (Bob Bond), Том Раппат (Tom Rappath), Джон Ловитт (John Lovitt), Кевин Хаар (Kevin Haar), Марк Сэдлер (Mark Sadler), Джон Лэмберт (John Lambert) и Бертон Голдфилд (Burton Goldfield). Эти люди в совершенстве владели своим предметом. У Джона Ловитта я перениал перспективный взгляд на то, как должна создаваться организация. Боб Бонд был, вероятно, лучшим наставником, какого можно себе представить. А находившийся в тени Роберт Герстен (Robert Gersten) был мне почти братом.

В Rational со мной работали самые потрясающие разработчики программного обеспечения и менеджеры: Джим Арчер (Jim Archer), Джек Тилфорд (Jack Tilford), Дэйв Бернстейн (Dave Bernstein), Дэйв Стивенсон (Dave Stevenson), Рич Рейтман (Rich Reitman), Хауард Ларсен (Howard Larsen), Ашиш Викрам (Ashish Vikram, теперь в Бангалоре), Скотт Джонсон (Scott Johnson), Фил Гаррисон (Phil Garrison), Пит Стейнфелд (Pete Steinfeld) и Тим Кейт (Tim Keith). Каждый из них открыл мне что-то такое в разработке программ, что изменило мой взгляд на эту область. Джим несколько сар-

кастически познакомил меня с идеей «управления, которое не зависит от содержания», а Джек учил, что никогда и ни за что нельзя сдаваться.

Стокгольм: Яаак Урми (Jaak Urmi), учивший меня тому, как крупные организации реагируют на изменения, происходящие в технологиях.

Вот те люди из Rational и других организаций, которые рецензировали первоначальные варианты этих глав и сделали существенные предложения, позволившие их улучшить: Филипп Крухтен (Philippe Kruchten), Паскаль Леруа (Pascal Leroy), Уокер Ройс (Walker Royce), Гради Буч (Grady Booch), Дэйв Бернстейн (Dave Bernstein), Макс Видеман (Max Wideman), Пол Кэмпбелл (Paul Campbell), Боб Стейнбергер (Bob Steinberger), Джек Тилфорд (Jack Tilford), Джим Арчер (Jim Archer), Рич Рейтман (Rich Reitman), Дэвид и Марк Мараско (David and Marc Marasco) и Кэйт Джонс (Kate Jones). Никакие похвалы не будут излишними в отношении этой прекрасной группы рецензентов, не давшей мне соврать.

В Rational отмечу своих редакторов из «The Rational Edge», Майка Перроу (Mike Perrow) и Марлен Эллин (Marlene Ellin). Как у джентльмена не может быть секретов от своего дворецкого, так и у меня нет писательских секретов от них. Они классно сыграли свои роли.

Вот рецензенты рукописи, уже попавшей в издательство Addison Wesley: Марти Лессер (Marty Lesser), Гэри Поллис (Gary Pollice), Джон Уокер (John Walker), Стив Франклин (Steve Franklin), Борис Люблинский (Boris Lublinsky), Джоакин Миллер (Joaquin Miller), Карл Вигерс (Karl Wieggers), Дон Грей (Don Gray), Билл Ирвин (Bill Irwin), Джерри Рудизин (Jerry Rudisin) и Боб Бонд (Bob Bond). Кроме того, Первый Аноним и Второй Аноним.

В Addison-Wesley: Мэри О'Брайен (Mary O'Brien), Крис Зан (Chris Zahn), Бренда Маллиген (Brenda Mulligan), Майкл Терстон (Michael Thurston), Эйми Хассос (Amy Hassos, Specialized Composition, Inc.) и Лиза Тибо (Lisa Thibault). Благодарен им всем за смелость опубликовать не совсем обычную книгу.

Марвин Хошино (Marvin Hoshino), мой шурин и верный соратник, который много лет назад редактировал мою докторскую диссертацию и не получил за это признательности или благодарности. Тим Бреннан (Tim Brennan), мой верный сотрудник и финансовый директор в бытность мою старшим операционным вице-президентом в Rational. Кэрол Вайсс (Carol Weiss), столько сделавшая за эти годы для моей жены и семьи. Все они помогли мне остаться в здравом уме.

Вот ряд авторов, вдохновивших меня своими книгами. Их труды повлияли на то, что и как я пишу: Ральф Пальмер Агнью (Ralph Palmer Agnew), Форман С. Эктон (Foreman S. Acton), Ричард У. Хемминг (Richard W. Hamming), Ричард П. Фейнман (Richard P. Feynman), Морис д'Окань (Maurice d'Ocagne), Фредерик П. Брукс (Frederick P. Brooks), Михай Чиксентмихайи (Mihaly Csikszentmihalyi), Гради Буч (Grady Booch), Филипп Крухтен (Philippe Kruchten) и Уокер Ройс (Walker Royce).

Билли и Лестеру[”] за музыкальное сопровождение. Р. П. Фейнману, ставшему живым доказательством того, что мальчишка из Фар-Рокауэя может добиться успеха.

Все они показали, что в одиночку далеко не уйдешь и что возможность работать с такими замечательными людьми — это настоящая удача.

Джо Мараско
Пebbл Бич
Декабрь 2004

[”] Билли Холидей (Billy Holiday), джазовая певица, и Лестер Янг (Lester Young), легендарный саксофонист.



Предисловие

Почему следует прислушаться к тому, что Джо Мараско пишет о разработке ПО и людях, которые этим занимаются?

Весной 1991 г. мы отметили пятилетие работы Джо в компании специальной наградой. В те времена в Rational Software работало не так много людей, такие награды изготавливались индивидуально, и для Джо была придумана достаточно необычная вещь. Мы достали фигурку бульдога, которыми украшают капоты грузовиков Mack, и закрепили ее на пластинке. Все единогласно решили, что эта награда символизировала целеустремленность и цепкость, проявляемые Джо при достижении поставленных целей.

Поэтому неудивительно, что немного позже мы выбрали Джо руководителем проекта, ставшего для Rational переломным. В то время наш лидирующий продукт, Rational Environment, работал на специальном аппаратном обеспечении, и мы сочли необходимым перевести его на платформу UNIX. Это было неизбежно, но связано с риском: попытки перенести свой продукт со специализированной платформы на стандартную привели к гибели не одну фирму, и этот путь усеян их останками, в частности там полегла Daisy, специализировавшаяся на автоматизации проектирования электроники (Electronic Design Automation – EDA) и ориентированная на один язык – Symbolics. Ясно было, что задача сложная, исход неизвестен, а решать ее нужно.

При этом Джо по стандартам Силиконовой долины явно относился к патриархам: ему было уже 46. Но мы верили, что его опыт и твердая рука

проведут проект через все трудности. И мы знали, что Джо сделает все необходимое для успеха.

Последующие события говорят сами за себя. В сентябре 1991 г. Джо возглавил новую команду, которой предстояло через два года выпустить «Rational II» для двух платформ UNIX. Через семь месяцев был создан и работал прототип с ограниченным набором функций. Через 16 месяцев группа смогла продолжить разработку, базируясь уже на частично реализованном продукте. Он был разработан для двух UNIX-платформ – IBM и Sun, получил название Rational Apex и вышел точно в срок – через два года после начала работ, как и было обещано.

Проект Apex оказался чрезвычайно успешным, и клиенты пользуются им до сих пор. Джо управлял этим проектом, в том числе во время выпуска версий 2.0 и 3.0, а также руководил его переносом на все платформы UNIX и на платформу Windows. Важно отметить, что в течение 10 лет, следовавших за выходом Apex 1.0 в 1993 г., Джо выступал в роли спасителя, когда компания испытывала трудности с выпуском продукта. По мере роста компании в результате слияний и поглощений за Джо закрепились роль специалиста по выходу из чрезвычайных ситуаций, приходящего на помощь в самом нужном месте в самый тяжелый момент.

Одна из причин успеха Джо в обеспечении своевременной поставки продукта заключалась в том, что он проводил массу времени с разработчиками, вникая во все детали продукта и проблем его создания. Но он также много общался с клиентами Rational и хорошо знал их нужды. Поставка продукта всегда оказывается результатом достижения многочисленных компромиссов, и Джо неизменно располагал информацией, позволявшей ему принимать правильное решение о том, в каком направлении должен развиваться продукт.

Ближе к концу своей карьеры в Rational Джо начал публиковать статьи о разработке ПО в электронном журнале компании, «The Rational Edge». В отличие от статей, принадлежавших перу трех «амигос» («Three Amigos»), его материалы были ближе к практике и основывались на опыте, полученном им в Rational и предшествующих местах работы. Оказалось, что Джо умеет убедительно изложить свой опыт и стать «виртуальным наставником» начинающих менеджеров программных разработок со всего света. Реакция на эти статьи была весьма положительной, и приятно видеть, что теперь все они собраны в одной книге.

Эта книга не теоретическое исследование процесса разработки ПО. Этим Джо заниматься не собирался. Он поставил цель помочь коллекти-

вам разработчиков организовать работу так, чтобы создаваемыми продуктами можно было гордиться, чтобы их было легко сопровождать и чтобы они были действительно полезны клиентам. Если вы хотите разрабатывать продукты, которыми можно гордиться и которые приносят пользу клиентам, вам надо обязательно прочесть эту книгу.

Майк Девлин (Mike Devlin),
главный управляющий Rational software, IBM



Введение

Эта книга в значительной мере основана на серии статей под названием *Franklin's Kite* (Воздушный змей Франклина), появившейся в электронном журнале Rational Software «The Rational Edge» в начале этого века. Они предназначались руководителям в области разработки ПО и преследовали цель предостеречь читателей от ловушек, в которые часто попадают программные проекты и разработчики. Статей вышло более 20, и мы – мой редактор Майк Перроу (Mike Perrow) и я – обратили внимание, что многие начинают чтение ежемесячного выпуска журнала именно с этой серии.

Я хотел не просто собрать эти статьи вместе, но и организовать их так, чтобы они принесли еще большую пользу руководителям программных разработок *и их руководителям*. С этой целью я расположил их по темам, а не в порядке появления на свет. При этом пришлось немного отредактировать их, чтобы избежать появления «ссылок вперед». Я также поработал над сносками, которые в оригинале часто представляли собой URL, а в книге превратились в более формальные цитаты, когда это было уместно. Наконец, я сделал добавления в начале и конце каждой главы, благодаря которым стал яснее контекст каждой статьи как части целого.

Читатель быстро обнаружит присутствие нескольких разных стилей изложения материала. Одни главы описательные, другие скорее аналитические, а третьи представляют собой некие «беседы с Сократом», в которых автор вступает в диалог с мифическим персонажем по имени Роско Леруа. Этот Роско воплощает тип хорошего технического менеджера, который прежде не занимался разработкой программ. Он играет роль парт-

нера, «неискушенность» которого вынуждает меня избегать технического жаргона. Я не пренебрегаю никакими средствами, чтобы передать свою мысль. Одним читателям понравятся главы в таком стиле, другим – в ином. Что даст нужный эффект, то и правильно. Я взял пример с Горация, написавшего в «Науке поэзии»: «Всех голоса соединит, кто мешает приятное с пользой, и занимая читателя ум, и тогда ж поучая».¹

Книга разделена на шесть частей, по четыре главы в каждой. Вот их темы:

- **Общий менеджмент.** Эти главы содержат материал, полезный менеджерам в целом, а также знакомят читателя с моим опытом и пристрастиями. Я включил их в качестве основания, на котором строится дальнейшее изложение.
- **Особенности разработки ПО.** В этой части мы посмотрим, какие особые задачи управления возникают при разработке ПО.
- **Разработка ПО с точки зрения управления проектами.** Здесь я рассматриваю **программный** проект как разновидность проекта вообще, благодаря чему возможно применение классических методов управления проектами. В то же время я стараюсь подчеркнуть специфические отличия разработки программ.
- **Человеческий фактор.** В этой части я захожу с другой стороны и рассматриваю разработку ПО с точки зрения того, кто практически осуществляет ее. И здесь я снова стараюсь сравнивать и противопоставлять, выявляя сходства и различия между программными проектами и всеми прочими.
- **Нестандартный подход.** Есть много разных точек зрения на решение проблем разработки ПО. В этой части я излагаю некоторые рискованные и неожиданные идеи, с которыми читатель, возможно, не встречался.
- **Более сложные вопросы:** Успешного администратора программных проектов можно сравнить с удачливым игроком в пинбол: наградой за набранные очки служит дополнительная бесплатная игра. А эти новые игры ведут к еще большему росту его мастерства. В этой части я поговорю о ряде проблем, возникающих с приходом успеха.

¹ Перевод М. А. Дмитриева. То же самое в прозе: «Общего одобрения заслуживает тот, кто соединил приятное с полезным». Оригинальный латинский текст Горация: «Omne tulit punctum qui miscuit utile dulci, lectorem delectando pariterque moneendo». Строка 343 в «Ars Poetica».

В этой книге 24 главы.¹ Можно читать их подряд, а можно выбирать произвольно: они достаточно независимы. Это неплохая книга «для самолета»: прочтите какую-нибудь главу, а потом размышляйте над ней до конца рейса. Если вы найдете хотя бы одну новую идею в какой-нибудь из глав, которая окупит для вас стоимость книги, я сочту свой труд успешным.

Покончив с этими предварительными замечаниями, приступим к делу.

¹ В «Илиаде» их столько же, но это случайное совпадение.

ЧАСТЬ ПЕРВАЯ



Общий менеджмент

Первая часть книги посвящена вопросам, представляющим общий интерес.

Глава 1 «В начале начал» знакомит с темой разработки ПО и моими личными взглядами на управление им. Я делюсь некоторыми мыслями по поводу общих проблем менеджмента.

В моих взглядах отразились полученные мной образование и опыт, поэтому в главе 2 «Мои вычислительные корни» я рассказываю о себе немного подробнее. В частности я показываю, что мой подход к разработке программного обеспечения глубоко укоренен в «расчетах», т.к. в начале своей карьеры, когда вычислительные науки только зарождались, я занимался программированием научных задач, и моей «базовой подготовкой» было инженерное дело.

В главе 3 «Альпинизм» я провожу аналогию между альпинизмом и проектом разработки ПО. В молодости я немного занимался этим видом спорта и пришел к выводу, что существует масса параллелей между этими двумя видами деятельности и командами, которые ими занимаются.

В главе 4 «Менеджмент» я рассуждаю о том, как должен (или не должен) поступать хороший менеджер.

ГЛАВА ПЕРВАЯ

В начале начал



Несколько лет назад мой кардиолог сообщил, что мне надо имплантировать кардиодефибриллятор (implanted cardioverter defibrillator – ICD). У меня заболевание, называемое желудочковой тахикардией, при которой сердце временами начинает биться слишком часто. Это может привести к фибрилляции, обычно приводящей к смерти. Устройство размером с колоду карт следит за ритмом сердца и генерирует электрический разряд, если обнаружит слишком длительное нарушение ритма сердечных сокращений. Этот разряд аналогичен тому, который получают с помощью хорошо известных электрических пластин при оказании срочной медицинской помощи.

Установка ICD – не простая процедура. Как опытный руководитель проектов я быстро провел анализ рисков. Вопрос формулировался в двоичной системе: устанавливать или не устанавливать? Можно и не устанавливать, если исходить из предположения, что фибрилляция никогда не произойдет, и тогда это будет правильным решением. Но если фибрилляция все же случится, то отказ от установки ICD скорее всего станет ошибочным и роковым «решением». Эта ветвь дерева решений склоняла меня к тому, чтобы согласиться на операцию.

Но были и другие факторы. Хотя риск, связанный с самой операцией, довольно мал, существует вероятность отказа устройства в нужный момент – не лучше ли тогда вообще его не иметь? Еще больший интерес представляет возможность ошибки «второго рода», т. е. срабатывания устройства в отсутствие реальной угрозы фибрилляции.

Одно не вызывало сомнений: при срабатывании устройства пациент испытывает то, что мой врач назвал словом «неудобство». Конечно, умереть из-за неполученного электрического разряда – это еще большее «неудобство», но легко представить себе, что и ложное срабатывание устройства несет угрозу жизни. Эту ветвь дерева решений также надо было обдумать.

Где мог возникнуть отказ? Само устройство достаточно простое. Есть батарейка с длительным сроком службы, конденсатор большой емкости и микропроцессор с программным обеспечением. Батарейка и конденсатор – это стандартные технические детали, которые могут быть подвергнуты обычному контролю качества; то же относится и к микропроцессору. Но что можно сказать о программном обеспечении?

Большая часть моей профессиональной деятельности была связана с управлением проектами разработки программного обеспечения. Я слишком хорошо знал, как разрабатываются программы, и потому меня обеспокоило, что 24 часа в сутки моя жизнь будет зависеть от некоего программного кода. Сколько бы мне ни оставалось жить, все это время в мою сердечную мышцу будут вживлены электроды, способные передать ощутимый электрический разряд по команде программы, наблюдающей за ритмом сердца.

Я все же решился на операцию, и вот уже семь лет, как мы с этим устройством успешно сосуществуем. Данные, которые можно прочесть из памяти устройства, показывают, что оно несколько раз собиралось подать разряд, но реально сделало это лишь однажды. Да, это было «неприятно», но не слишком. Я счастлив, что оно сработало, когда это потребовалось.

Значение качества программ

Этот пример непосредственно иллюстрирует, какую важность имеет разработка ПО. Для меня и тысяч других людей эта программа «критически важна». В большинстве встроенных приложений можно не обращать внимание на мелочи. Если программа, управляющая СВЧ-печкой, иногда допускает ошибки, то в крайнем случае вы что-нибудь пережарите в ней. Но программа, управляющая системой ABS для тормозов в машине должна работать всегда, как и прочие программные системы в транспортном средстве – управляющие работой двигателя, сцеплением с поверхностью дороги, вентиляцией салона, показаниями на приборной доске и т. д. Организовать правильное функционирование таких взаимозависимых систем не просто. А по мере увеличения количества взаимодействующих

распределенных программных систем вероятность возникновения непредвиденных ошибок растет.

Мы хотели бы доверять программам, которые во все большей степени управляют окружающим нас миром, но, если только вы не работаете сами в отрасли, занятой разработкой ПО, едва ли вы что-то знаете о том, как создаются, тестируются и эксплуатируются программы. Качество ПО бывает очень разным. Остается только надеяться, что чем более критически важным является приложение, тем выше качество соответствующего ПО, иначе может произойти катастрофа.

За последние 20 лет профессионалы в области разработки программного обеспечения существенно продвинулись в том, чтобы ввести в эту отрасль «инженерную дисциплину». Обычно это означает создание и неуклонное применение более совершенных «технологий». Это хорошо, но не решает проблему полностью. Отсутствие технологии может погубить проект, но одно лишь наличие хорошей технологии еще никогда не приводило к появлению хорошего ПО.

Для создания хороших программ нужны хорошие работники. И не менее важно управлять ими с умом. На разработчиков ПО – «программистов» – всегда валятся все шишки: какие бы не возникли неприятности, во всем обвиняют их. На самом деле неудачи программ обычно оказываются результатом недостаточной увлеченности программистов в сочетании с небрежным управлением. Эта комбинация приводит к появлению программ, которые работают, но не всегда и не очень хорошо.

Острова надежности

Я основываюсь на личном опыте, включающем в себя 30 лет работы в области управления программными проектами и 10 лет службы рядовым, которые ей предшествовали. За этот период в нашей отрасли почти всюду произошли глубокие перемены, коснувшиеся аппаратной базы, операционных систем, сетевого взаимодействия, языков, инструментов и т. д. Можно обоснованно утверждать, что на сегодняшний день разработка программного обеспечения – самая бурно развивающаяся область техники. Поэтому я попытался выяснить, что же осталось неизменным за эти 40 лет. Ибо если за эти суматошные 40 лет что-то не изменилось, то не исключено, что оно не изменится и за следующие 40. Информация об островах надежности в этом хаосе поможет нашему продвижению вперед с учетом того, что область продолжит свой рост, развитие, подвергаясь мимолет-

ным капризам и увлечениям моды. Хорошие администраторы по-прежнему будут добиваться успеха, соединяя «вечные истины» с новейшими технологиями, отыскивая правильное соотношение между консерватизмом прошедшей проверки временем практики с одной стороны и риском и вознаграждением, связанными с новшествами, с другой.

Добрая половина моей карьеры связана с Rational Software, ставшей теперь одним из пяти брэндов IBM. Влияние этой компании на образ моих мыслей было очень глубоким, и я много приобрел там для себя в трех отношениях.

Во-первых, мне пришлось сотрудничать со многими очень непростыми клиентами в разных странах мира, которые вели очень крупные и сложные проекты разработки ПО в целом ряде различных прикладных областей. Благодаря этим клиентам я получил представление о ключевых факторах успеха или провала проекта, а это крайне ценные данные.

Во-вторых, мне довелось руководить рядом совершенно замечательных коллективов, и если мне удавалось добиться какого-то успеха в качестве руководителя программного проекта, то в значительной мере я обязан этим таланту людей, которыми мне довелось руководить. Хороший управляющий может повысить эффективность работы отличной команды, но, как мне однажды сказал знающий завсегда тайп ипподрома, «я еще не видел жокея, который тащил бы свою лошадь через финишную прямую».

Наконец, мне посчастливилось сотрудничать с некоторыми действительно выдающимися мыслителями, которые не только понимали, что и как действует, но и не ленились тщательно объединить свой опыт в одно целое и ясно изложить его. В этой компании было постоянное движение, крутился водоворот подчас конкурирующих между собой теорий и теоретики, методисты и прагматичные администраторы присутствовали на настоящей ярмарке идей. Должен признаться, что ни один из моих коллег ни разу не сказал: «Ладно, Джо, может быть, это и действует на практике, но что говорит по этому поводу теория?». Нет, я склонен считать, что мы достигали целей благодаря тому, что успешно соединяли новые теоретические подходы с трудно доставшимся практическим опытом.

Это не значит, что я все еще был новичком, когда пришел в Rational в 1986 г. У меня на шкуре было уже немало шрамов, приобретенных в войнах программных разработок. Мне довелось повидать хорошее, плохое и ужасное, и у меня уже были свои взгляды. Rational собрала группу талантливых специалистов, которые составили настоящую дружную команду и стремились сделать результаты своей работы полезными для клиентов.

Мое участие в этом предприятии в течение более чем 16 лет было для меня большой радостью.

Для кого предназначена эта книга

Эта книга может пригодиться не только менеджерам программных проектов, но и *их* менеджерам, которые обычно очень мало знают о программном обеспечении. Весьма опасно, когда старший по должности выдвигает (по своему неведению) неразумные требования, а управляющий программным проектом уступает ему (со слишком большой готовностью), тогда как не должен был этого делать. В результате через год выпускается рыхлый продукт. Еще через год все тычут в него пальцем, а программистов клянут за плохую работу. Ошибочное решение!

Поясню свою позицию простыми словами. Чтобы быть хорошим менеджером программного проекта, надо прежде всего быть хорошим менеджером. Большинство общих принципов управления производственными коллективами применимо к коллективам разработчиков ПО. Точно так же применимо и большинство обычных, стандартных принципов управления проектами. Если вы хорошо разбираетесь в программировании, но нигде не ходите как менеджер или нарушаете основные принципы управления проектами, вас ждет неудача. Вот почему значительная часть того, что написано в этой книге, очень напоминает «общие принципы управления». Эти базовые идеи должны лежать в основе вашей деятельности.

Еще надо понимать, в чем заключаются особенности разработки ПО. Если предыдущий абзац адресован новичку в области управления программным проектом, то этот адресован его боссу. Да, дорогой босс, вы отличный администратор широкого профиля, потому вы и стали начальником. Но вы должны знать, что у программного обеспечения есть свои особенности. Они касаются широкого круга вещей и отчасти вызваны тем, что соответствующая научная дисциплина еще относительно молода и незрела. Чем больше вы узнаете об этих естественных отличиях, тем более осмысленным будет ваше общение с подчиненным вам менеджером программного проекта. А это означает, что вдвоем вы сможете принимать более компетентные решения в отношении разработчиков. Прочтя эту книгу, вы узнаете, в чем состоят особенности разработки ПО. Не заваливая вас техническими словечками, мы постараемся, чтобы вы как участник процесса стали лучше разбираться в предмете. Потраченные вами время и труд должны принести немалые дивиденды.

Циферблат для итеративного решения задач

Кем бы вы ни были – менеджером программного проекта или менеджером этого менеджера – вам придется каждый день решать целый ряд практических задач. Как решать задачи? Будет ли результат случайным – иногда успешным, а иногда совершенно плачевным? Случается ли вам «застрять» на одном месте? Сложно ли вам выбрать направление приложения своей творческой энергии и трудно ли положить конец спорам?

Все мы иногда сталкиваемся с подобными трудностями. Однако есть способ управлять этим очень важным процессом. Я решаю задачи путем как минимум двукратного обхода круга, показанного на рис. 1.1, а при необходимости повторяю его снова. На практике процесс можно начать почти в любой точке круга, но для удобства я начну свое изложение в самом «логичном» месте – на 9 часах. И вместо «я» буду теперь говорить «мы», потому что мы проделаем все *вместе*!

Первый циклический обход

Этапы, которые мы проходим во время первого цикла, немного отличаются от тех, которые будут потом:

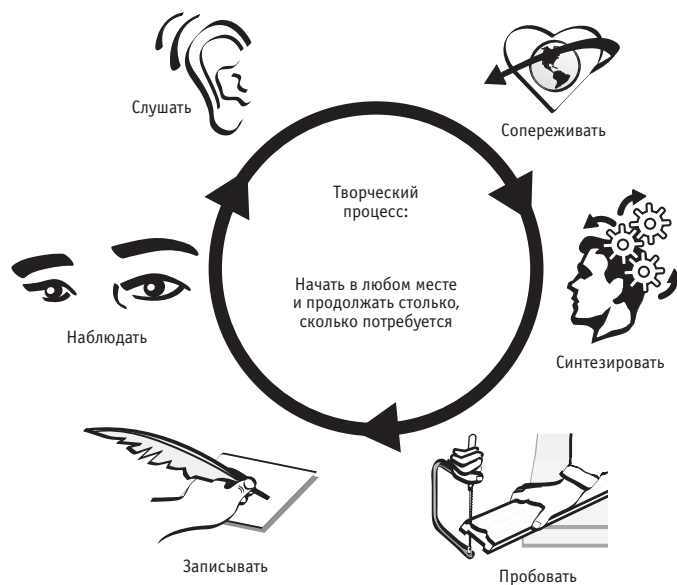


Рис. 1.1. Циферблат для итеративного решения задач

- **Девять часов: наблюдать.** Надо быть *в курсе*. Чутье должно действовать постоянно. Мы напряженно наблюдаем за всем, что происходит вокруг, и стараемся обнаружить проблемы.
- **Одиннадцать часов: слушать.** Выявив проблему, советуемся с другими и выясняем, что они думают. Слушаем активно, ведем сократовские диалоги с партнерами, чередуя роли учителя и учащегося. Стараемся больше слушать, чем говорить, и тщательно все записываем.
- **Час дня: сопереживать.** Я отделил это от действия «слушать», чтобы вы почувствовали разницу. Можно слушать, чтобы получить «объективные» данные; *сопереживание* – это мостик между слушанием как сбором фактов и началом синтетической стадии процесса. Мы слушаем ушами, а синтезируем мозгами; в сопереживании участвует сердце. Для этого необходимо забыть про себя и поставить себя на место другого человека. Это не всегда просто сделать, особенно если то, что мы слышим, не согласуется с нашими прежними представлениями. Поэтому синтезу должно предшествовать сопереживание. Менеджеры, прежде работавшие в машиностроении, иногда пропускают этот шаг, потому что привыкли решать проблемы, природа которых на 99,44% техническая; к сожалению, для большинства проблем общего управления это неверно.
- **Три часа: синтезировать.** Теперь соединяем все части вместе:
 - Данные, которые мы собрали, наблюдая и слушая
 - Эмоциональные аспекты, обнаружившиеся при взгляде на проблему с чужих точек зрения
 - Наш прежний опыт решения аналогичных проблем
 - Некий «посторонний» опыт, имеющий отношение к делу
 - Наш арсенал методов, процедур, стратегий, тактик, приемов и трюков
 - Придумываем пробное решение проблемы на основе всего, что нам известно.
- **Пять часов: проверяем его.** Важный шаг, на котором мы выполняем проверки и ставим эксперименты, чтобы выяснить, приводит ли предложенный путь к решению проблемы. Здесь отсеиваются те идеи, которые хорошо выглядели на бумаге, но не реализуемы в действительности. Этого не узнаешь, пока не попробуешь. Будьте особенно критичны к себе на данном этапе: отказ от скверных решений – это важная часть творческого процесса решения проблем.

- **Семь часов: все записываем.** Здесь мы документируем решение, а также выполненные проверки и эксперименты. В таком виде решение предстанет перед теми, кого оно касается, поэтому хорошенько потрудитесь над ним. Если не оформить его письменно, труднее будет убедить других в правильности ваших мыслей, и то, что не записано, часто быстро забывается.

Второй и последующие проходы цикла

Мы выработали пробное решение, и прошло достаточно времени, чтобы его могли оценить другие. Пора пройти цикл снова. На этот раз шесть этапов будут выглядеть так:

- **Наблюдать:** Посмотреть, как реагируют люди на пробное решение. Есть ли у нас потери?
- **Слушать:** Поговорить с людьми о плюсах и минусах решения. Пусть выскажутся, что им нравится, а что нет, что действует, а что не действует.
- **Сопереживать:** Какие факторы повлияли на то, что мы услышали? Может быть, люди просто сопротивляются переменам? Не задело ли решение кого-то лично? Нет ли каких-то побочных эффектов и нежелательных последствий, кого-то огорчивших? Или оно слишком противоречит здравому смыслу?
- **Синтезировать:** Полученные новые данные надо также ввести в процесс синтеза. Можно ли немного поправить первоначальное решение или требуется **его** основательно переработать? Как правило, можно двигаться дальше, модифицировав или усовершенствовав результат предыдущего прохода. Но не бойтесь начать синтез с нуля, если первая попытка оказалась неудачной.
- **Проверять:** С каждым новым проходом этого этапа нам становится все понятнее, как проверить решение. Ведь у нас есть результаты проверок от предыдущего цикла и новые данные о том, что должно быть проверено. Подойдите к своему решению жестче, чем возможные критики. Готовьтесь к возражениям и проблемам и смотрите, как поведет себя ваше решение, столкнувшись с ними.
- **Записывать:** Взяв в качестве отправной точки первоначальный документ, измените **то**, что необходимо, и укажите, что именно было изменено или усовершенствовано.

Когда считать проблему решенной?

Когда можно остановиться? Подвести черту очень важно: мы вовсе не хотим крутиться в этом цикле вечно. Вот неплохой ориентир – остановиться надо в том случае, если на этапах наблюдения и слушания получено очень мало существенной новой информации.

Процесс можно начать в любой точке окружности. Решая проблемы, надо проявлять гибкость. Реальная жизнь проходит «беспорядочно», и иногда идея зарождается во время какого-нибудь случайного синтеза. На первый взгляд, лучше остановиться и вернуться обратно «на 9 часов», но иногда лучше воспользоваться моментом и двинуться дальше с того же места. Не забывайте, что независимо от начальной точки надо проделать хотя бы два полных цикла, чтобы гарантировать обратную связь. Вот почему так важен этап «семь часов»: он обеспечивает обратную связь.

Обратите также внимание на очень полезный побочный эффект: поскольку мы ведем записи при каждом прохождении цикла, совершаемом не менее двух раз, в конце нам приходится значительно меньше заниматься канцелярской работой. «Окончательный документ» создается последовательно как органическая составная часть итеративной выработки решения.

Выработка решений с помощью такого метода имеет тенденцию «вязнуть». Чтобы избежать этого, надо проходить все ступени жестко и с рассчитанной скоростью. Не пропускать шагов и не застревать слишком долго ни на одном из них. Приобретя достаточный опыт, можно начать импровизировать. Но вначале соблюдайте описанные рамки и обуздывайте свои творческие порывы. Возможно, результаты приятно удивят вас.

Резюме

Можно представить себе эту книгу как результат того, что в различных ситуациях я останавливался на семи часах и «записывал». Книга представляет собой некоторую совокупность остановок, во время которых мне удалось законсервировать временные решения проблем, которыми я тогда занимался. Когда настал момент собрать их и объединить в книгу, мне пришлось сделать еще один проход. Я обнаружил, что в большинстве случаев записанное мною требовало незначительных изменений. Результат, устойчивость которого была обеспечена рядом предшествующих и последующих итераций по кругу, прошел проверку временем.

ГЛАВА ВТОРАЯ

Мои вычислительные корни



В разработку программного обеспечения я пришел с черного хода. В те времена, когда я учился в старших классах и колледже, область знаний «вычислительная техника» только зарождалась. Большинство менеджеров моего возраста, занимающихся разработкой ПО, также начинало в какой-то другой области: математике, физике, химии или машиностроении. О компьютерах мы узнали, когда они появились у нас в качестве нового инструмента для вычислений. После этого наше увлечение наукой или инженерным делом в некоторых случаях перешло на программирование. Со временем из этого возникло желание создавать программы высокого качества.

Я хочу сказать, что подхожу к проблемам управления разработкой ПО с иной точки зрения, чем те, кто получил формальное университетское образование на факультете вычислительной техники. Я склонен рассматривать эти проблемы в первую и главную очередь как проблемы управления инженерными разработками. Мне посчастливилось работать с несколькими действительно талантливыми программистами, и сочетание таланта и умений, которые мы вместе приносили в программный проект, оказывало на него очень сильное воздействие.

Прежде всего, как случилось, что я стал инженером? Это долгая личная история, которая может показаться вам интересной. Если нет, переходите к следующей главе, но сначала прочтите «Резюме» в конце этой.

Ниспровергатель

4 октября 1957 г. Советский Союз вывел на орбиту «спутник». Сообщалось, что размером он всего лишь с баскетбольный мяч, весит 184 фунта¹ и посылает сигналы, которые можно поймать коротковолновым приемником. Мир был в шоке, и в обществе бурно обсуждался удар, нанесенный превосходству Америки в области науки и техники. Все усугубилось тем, что 3 ноября того же года Советы запустили «Спутник II» весом 1121 фунтов,² на борту которого была собака по имени Лайка. Это произошло, когда первый спутник еще не успел пойти по спирали к Земле, чтобы прекратить свое существование, и эффект был огромен. Подобно бомбе, взорванной в Нагасаки после Хиросимы, Лайка показала, что русские могут делать такие вещи на регулярной основе, а не в виде единичного события. Пусть интервал между запусками специально был рассчитан на такой эффект, но психологически это было воспринято острее, чем соответствовало реальности. Холодная война была в разгаре, и русские сильно побили нас. Ведь если они смогли послать в космос собаку, не последует ли за ней вскоре обезьяна? А если уж там окажется обезьяна, не трудно видеть, что будет дальше. Запуск ракет из космоса становился вполне реальным. Говоря военным языком, русские заняли «командную высоту», а наше руководство в области науки, обороны и политики промахнулось.

Конечно, советский спутник оказался громом среди ясного неба, но это событие имело более важные последствия для страны. Джон Кеннеди воспользовался лозунгом «отставания в ракетной технике», который помог ему выиграть президентские выборы 1960 г., и затем вовлек страну в гонку в космосе 1960-х г., кульминацией которой стала высадка человека на Луне в 1969 г. За каких-то 12 лет мы догнали и обошли русских, вызвав трепет во всем мире. Это была яркая демонстрация силы «американского духа».

Конечно, потребовавшаяся для этого физика разрабатывалась еще Ньютоном, а создание ракетной техники, как ни странно, происходило под руководством ветеранов, создававших в Германии V-2 (если верить расхожему мнению). Но спутник, запущенный Советами, оказал влияние на целое поколение более скрытым образом: он бросил вызов самым основам нашей системы образования.

¹ 83,46 килограмма.

² 508,48 килограмма.

В принципе из запуска спутника следовало, что мы, американцы, отстали потому, что система образования у русских оказалась совершеннее и они смогли привлекать в науку и технику больше лучших и ярких умов. В то время как американцы оставались непревзойденными в массовом производстве товаров потребления и подъеме уровня жизни среднего класса, русские решали по-настоящему трудные проблемы и оказывались впереди нас в действительно важных областях. Неважно, так ли это было на самом деле, но такое ощущение было широко распространено и принималось безоговорочно. Какое-то время казалось, что вся страна пришла в смятение в связи с этими событиями. Что-то нужно было предпринимать.

Ответ

В Америке существует проверенный временем способ разрешения всех кризисов. Если вы еще не догадались, о чем идет речь, то скажу, что он называется «закидать проблему деньгами». Смысл в том, что можно в значительной мере избавиться от существующей в демократическом обществе необходимости заключать сделки и достигать соглашения, если воспользоваться для влияния на поведение людей системой поощрений, имеющихся в свободном предпринимательстве. Этот метод не всегда самый эффективный и почти всегда выглядит непривлекательно, но он бывает действенным. Он может привлечь массу ненужных людей, которые заинтересованы в быстром обогащении, а не в решении проблемы, но он также находит и много «нужных» людей.

Далее. Заинтересовать надлежало не молодых людей моего возраста, а другой контингент. Конечно, кого-то из нас естественным образом могли привлечь романтика, таинственность и глубина науки. Другие могли проявить интерес из патриотических чувств. Но в целом гормоны диктовали то, что для большинства из нас (я имею здесь в виду лиц мужского пола и юношеского возраста) основные «проблемы» в течение ближайших лет будут связаны с противоположным полом. Те из нас, кто был склонен к технике, видел здесь более серьезные и сложные проблемы, чем полет на Луну. Можете мне поверить, что вид привлекательных девчонок, сидящих на плечах атлетически сложенных мужчин, был не менее сильным средством мотивации, чем атомная энергия. Спутник ничего не изменил в этой психологии.

Нет, чью психологию надо было изменить, так это психологию наших родителей. Потому что период с конца 50-х по начало 60-х годов XX века был последним, когда дети хотя бы делали вид, что слушаются родителей.

Время было на редкость удачным: не позднее чем через 10 лет бунтарский дух поколения шестидесятых сведет влияние родителей до крайнего минимума. Прежде всего потому, что мы были буквально последним поколением, которое хоть сколько-нибудь прислушивалось к словам своих родителей. Трудно представить себе, чтобы тинэйджеры семидесятых и восьмидесятых были похожи на своих сверстников конца пятидесятых и начала шестидесятых. Вспомните сериал «Father Knows Best»¹ и культуру среднего класса, по представлениям которой марихуану курили только джазовые музыканты, чтобы испытать вдохновение.

Конечно, мы понятия не имели, что бывает как-то иначе.

Как действовала эта программа

Итак, началась великая пропагандистская кампания. Неофициальное послание родителям молодых американцев звучало примерно так: «Наука и техника имеют важнейшее значение для страны». Особенно важно было не упустить в этой кампании часть, касающуюся техники. Это обуславливалось в то время (и сейчас тоже) представлениями о науке и технике в общественном сознании. Заниматься наукой сложно; заниматься техникой – обычное дело. В науке есть обаяние, техника связана с грязью. Наукой занимаются философы-аристократы; инженеры ближе к синим воротничкам. Только блестящим людям открыт путь в науку; инженером может стать практически каждый, проявив достаточно трудолюбия. Все это стереотипы. Но задумайтесь: если бы вам нужно было с помощью национальной пропагандистской кампании привлечь как можно больше молодых людей в технические области, как бы вы поступили? Зазывали бы только в науку? Нет, чтобы улов оказался больше, надо было раскидывать сети шире и подчеркивать, что нужны и ученые, и инженеры. В результате если из кого-то не получится ученого, то не исключено, что из него можно сделать инженера.² Каждому родителю, сомневающемуся в том, что его сын

¹ «Father Knows Best» (Папа на все руки) – американский телевизионный ситком (от английского выражения situation comedy) того времени, изображавший «типичную американскую семью», в которой каждый представлял собою идеал, а все проблемы решались не более чем за 30 минут с учетом времени на рекламу.

² Внимательный читатель заметит, что это подкрепляет ранее упомянутые стереотипы. Однако студенты, изучавшие физику, чаще переходили в инженеры, чем наоборот. Как вы увидите дальше, я был любопытным исключением из этого общего правила.

сможет стать доктором медицины, приятно думать, что стоматология – почетная (и доходная) профессия.

Те, кому была адресована эта кампания, пережили Великую депрессию 1930-х. Из того, что за ней последовала Вторая мировая война и в тот момент (1957 г.) у них были дети школьного возраста, можно заключить, что сами они приличного образования не получили. (Они не ходили в колледж из-за депрессии, они потеряли еще пять лет из-за войны в начале 1940-х, а после войны они женились и завели детей. Многие из них так и не воспользовались своими правами участников войны.) Теперь они стали рабочими и принадлежали к нижней части среднего класса, дети готовились поступать в старшие классы, и материальные условия были нелегкими. Реклама была придумана толково: пусть ваш ребенок станет ученым или инженером, и он будет обеспечен на всю жизнь.¹ «Это ваш счастливый шанс. Не опоздайте. Постарайтесь, чтобы ваш ребенок избежал вашей судьбы!»

Дальше реклама становилась еще убедительнее: «Мы вам поможем. Мы выделим на образование столько денег, что ваши дети смогут обучаться бесплатно. Стипендий и дотаций будет полно. Пусть цена вас не волнует. Если у вашего ребенка есть талант, страна даст ему образование, и ему никогда больше не придется ни о чем беспокоиться».

Должен сказать, что это была замечательная кампания, и она была успешной.

Какой оказалась реальность? Как обычно, хорошее переমেжалось плохим. Во многие научные и технические программы влили уйму денег. И сделали много чего еще: снижали налоги, процентную ставку и т. д. Спутник запустили, когда мне было 12 лет, а закончил я образовательную процедуру в 1972 г. (да, при серьезном подходе требуется много времени), когда получил докторскую степень по физике. Все мои занятия на старших курсах и в аспирантуре финансировались посредством заработков во время летних каникул и по выходным, стипендий и дотаций. Если не считать упущенных заработков, то следует признать, что мое образование не стоило мне (или моим родителям) ни цента. Я вышел на рынок труда полностью свободным от долгов. Только представьте себе – не надо отдавать никаких займов на образование!

¹ Местоимение мужского рода употреблено намеренно. Мы тогда еще не сообразили, что и женщины могут быть учеными и инженерами. Для этого потребовались еще 10 лет и еще одна революция.

Плохо оказалось то, что обещанная пожизненная похлебка оказалась не более чем иллюзией, как и всегда случается с такими обещаниями. Наверное, сейчас трудно представить, чтобы кто-то мог поверить, будто степень бакалавра по инженерному делу обеспечит его материально на всю жизнь. Но тогда эта возможность казалась гораздо более правдоподобной и практичной, чем другие, и множество молодых мужчин (а позднее и женщин) выбрали этот путь. В одном отношении он оказался верным. Например, в начале семидесятых оказалось, что у нас физиков хоть пруд пруди, и закономерно возник кризис занятости, поскольку предложение превосходило спрос (по крайней мере, в научных кругах).¹ Мы забыли выключить насос: как всегда в делах, где участвует правительство, о тонкой настройке и мечтать нечего.

Конечно, в 1960 г. мы и не догадывались, что такое может произойти. Это было так же невероятно, как высадка человека на Луне.

Что было особенного в этом поколении инженеров?

После этой довольно длинной прелюдии я хочу сказать несколько слов о 10 годах обучения студентов инженерных специальностей примерно с 1960 по 1970 г. Я поступил в инженерный колледж в 1962 г. и окончил его со степенью бакалавра химической технологии в 1966 г., поэтому считаю себя представителем указанной группы. После интенсивного обучения в старших классах я поступил в колледж почти ровно через пять лет после запуска спутника Советским Союзом. Я собирался стать инженером.

Кроме того, я собираюсь рассказать об инженерах, а не об ученых. Я выполнил свою дипломную работу по физике и получил докторскую степень по экспериментальной физике частиц высоких энергий в 1972 г., поэтому я могу претендовать на то, чтобы считаться и инженером, и физиком. Но в то время как в физике и других науках в это десятилетие произошло относительно мало изменений, в инженерном деле и инженерных колледжах изменения были радикальными.

Попутно замечу, что это, вероятно, была последняя крупная волна студентов-инженеров, *родившихся в Америке*. После того как прошла эта группа, мы снова перестали пускать своих детей в инженеры. Образовавшийся вакуум был заполнен студентами с других континентов.

¹ По иронии судьбы именно этот кризис занятости подтолкнул меня в сторону разработки программного обеспечения.

Это поколение студентов инженерных специальностей оказалось в точке перегиба. Мы были последними, кому преподавали некоторые классические инженерные дисциплины. В разных колледжах были свои особенности (Соорег относился к консервативным заведениям), но до 1960 г. дело обстояло одним образом, а после 1970 г. – иным. Шестидесятые годы были переходным периодом, и в зависимости от того, в каком колледже вы учились, пропорции преподаваемых дисциплин несколько различались.

Например, мой первый курс был одним из последних наборов студентов, которым преподавались техническое черчение и проективная геометрия. Черчение подразумевало вычерчивание технических чертежей профессионального качества при помощи измерителей, циркулей и угольников. Сегодня вам может быть знакома лишь одна разновидность таких чертежей, называемая «синьками», а мы должны были понимать, что такое «вид сверху», «вид сбоку» и как на их основе создавать вид под произвольным углом. И – особое проклятие для меня – нельзя было закончить курс, не сделав хотя бы одного чертежа тушью на веленовой бумаге.¹ Это было частью той традиции, когда инженер находился рядом с механическим цехом. Нужно было уметь делать чертежи, понятные рабочему-станочнику. Это была та часть инженерного дела, которая предполагала «засученные рукава и распушенный галстук». Чтобы быть членом бригады, надо было уметь читать и изготавливать технические чертежи, даже если вы носите белую рубашку и галстук, а станочник носит рубашку с синим воротничком и фартук. В каком-то смысле вы должны были стать равны (хотя бы на время), и овладение мастерством черчения входило в ученичество. Мысль о том, что можно стать инженером, не научившись чертить детали механизмов, была нелепа. А сегодня мои рассказы об этом вызывают смех. Но тогда все это было очень серьезно. Представьте себе, что вы получаете высшую оценку по интегральному исчислению, потому что умеете брать интегралы по частям, и испытываете смертельный страх вылететь из колледжа, потому что прежде чем вы закончите проклятый чертеж, тушь обязательно затечет под угольник.

¹ Строго говоря, веленовая бумага (vellum) – это даже не бумага. Но это и не пергамент; ее делают из телячьей кожи. Но если бы я не сказал «веленовая бумага», никто и не понял бы, о чем идет речь.

Вычисления

В самой большой степени «перелом» затронул технику вычислений. Наше поколение оказалось мостиком между логарифмической линейкой и компьютером. Сейчас объясню.

В шестидесятые годы карманный калькулятор все еще был предметом из обихода Флэша Гордона.¹ Первый настоящий калькулятор для инженеров, HP-35, появился на сцене в 1972 г. (Какое совпадение – я как раз заканчивал свою диссертацию. Выбор времени – это все!) Логарифмическая линейка сразу и безвозвратно погибла. Но до того момента логарифмические линейки были главным товаром для инженеров. Попросту говоря, на линейке выполнялись расчеты. Применять компьютеры для получения обычных результатов тогда было просто невыгодно. Компьютеры были ориентированы на пакетную обработку, и чтобы получить результат, надо было написать программу. На языке FORTRAN. Слишком большие накладные расходы для разовой задачи. Это сегодня можно запустить Excel, а тогда приходилось вытаскивать логарифмическую линейку.

Учиться работать с логарифмической линейкой вы начинали на первом курсе, если не раньше. Недостаточно было понять, как ей пользоваться, требовалось научиться делать это уверенно, точно и быстро. С помощью линейки приходилось высчитывать ответы на 50-минутных экзаменах по физике, химии и инженерному делу. Тот, кто не умел пользоваться ей как настоящим инструментом или не умел считать быстро, оказывался чужим на этом празднике жизни. Это было так же плохо, как клякса туши под угольником.

Преподаватели тоже не давали сильно расслабляться. Иногда мы получали какие-то баллы, если показывали, что понимаем, как сделать расчет, но ошиблись. Но в итоге обнаруживалось, что без правильного ответа не было и достаточных баллов для получения хороших оценок. Какая свежая идея – куча баллов за правильный ответ и немножко за неправильный! Но, как заметил однажды один старый желчный профессор, «инженерам платят за правильные результаты». Кстати, я говорил, что скорость тоже учитывалась?

¹ Еще одна историческая справка. Флэш Гордон появился у нас задолго до Люка Скайуокера. Любопытно, что его зловещим двойником стал Минг Беспощадный, а не Дарт Вейдер. В те времена значительно меньше обращали внимание на поллиткорректность.

Поэтому мы тренировались. Герман Биленко (Herman Bilenko), студент старшего курса электротехнической специальности, руководил клубом совершенствования навыков работы с логарифмической линейкой. Мы встречались за завтраком, он давал нам задачу, а мы соревновались, кто быстрее получит правильный результат. Как говорил Дэйв Барри (Dave Barry), «я ничего не выдумываю».

Тесное знакомство с числами

Тем, кто никогда не имел дела с логарифмической линейкой, скажу, что чаще всего требовалось вычислить дробь, и в числителе, и в знаменателе которой было не меньше чем по четыре множителя. «Ответ», который давала линейка, мог выглядеть как, скажем, «123». Где поставить десятичную точку и каков будет показатель степени (например, «на десять в шестой степени»), должны решить вы сами. Это важнейший момент. Чтобы получить правильный ответ, требовалось правильно сделать две вещи. Во-первых, определить, где поставить десятичную точку в окончательном результате, полученном из восьми или более множителей. Во-вторых, определить, следуя правилам научного формата записи, какой показатель степени будет у окончательного результата, исходя из показателей степеней всех сомножителей. Крайне легко было в этих вычислениях сместиться на единицу в показателе или в положении десятичной точки. И если правильный результат был 1,23, то всегда можно было по ошибке или по небрежности получить 0,123 или 12,3. На профессиональном языке это называется «ошибка на порядок».

Конечно, ошибка в десять раз – это очень много. Представьте себе, что вам начислили зарплату, ошибившись в 10 раз, и вам все станет ясно. А при работе с логарифмической линейкой очень легко ошибиться в десять раз. Из этого следовало, что еще до проведения вычислений надо было сообразить, какой примерно ответ должен получиться. Нужно было «знать», что ответ получится в районе «единицы», и, если на линейке получался результат 0,123 или 12,3, становилось ясно, что допущена ошибка, и приходилось возвращаться и искать ее. Таким образом, оценивать результат приходилось заранее, и обойтись без этого было невозможно.

Те, кто не овладел этим мастерством, вылетали из колледжа, потому что совершить ошибку было очень легко. Если у вас не было нюха на ошибочный результат, вам грозили неприятности.

Понимание этих моментов приводило к выработке некоторых любопытных стилей поведения. Как правило, вычисления проводились в несколько этапов, при этом результат предыдущего этапа подставлялся в некоторую новую формулу, и это делалось многократно. Поэтому ошибки, сделанные на ранних этапах, размножались далее, и, когда в итоге получался совершенно абсурдный результат, приходилось мучительно двигаться в обратном направлении к самому началу. Как следствие, у нас развилась маниакальная привычка проверять каждый промежуточный результат, прежде чем двигаться дальше. Мы сами для себя стали «контролем качества» вычислений, не разрешая продолжать вычисления, пока не были уверены, что результат примерно правильный. Развить в себе такой инстинкт в самом начале карьеры было очень полезно.

Естественно, что попутно мы научились очень неплохо выполнять вычисления в уме. Никто, конечно, не мог сравниться с легендарным Фейнманом,¹ но и мы были совсем не плохи. Иногда такого искусства оказывалось достаточно, чтобы вскружить голову студентке-гуманитарии. Для нас же это был еще один навык, необходимый для выживания.

Кстати, часто предлагалось выполнить вычисления только с точностью до трех значащих цифр. Достаточно ли этого было? Это означало ошибку примерно в одну тысячную. В большинстве физических экспериментов хорошей считается точность $\pm 1\%$. Поэтому, если у вас было три-четыре множителя в числителе и три-четыре множителя в знаменателе и все они имели точность не выше 1% , было бы заблуждением считать, что можно получить ответ с точностью одна тысячная. Ergo, в большинстве расчетов можно было смело пользоваться логарифмической линейкой.

Конечно, с появлением компьютеров все изменилось. Расчеты стали выполнять путем численной итерации уравнений – заменяя производные конечными разностями. И поскольку для получения результата надо было повторить тысячи шагов, ошибки могли незаметно накапливаться. Вот почему в компьютерах точность представления данных должна быть гораздо выше: на каждом шаге нужна очень большая точность, чтобы ошибки не накапливались. Но в конечном итоге результат никогда не бывает точнее входных данных. Многие либо вообще этого не понимают, либо с годами забыли об этом.

¹ Тоже закончил среднюю школу в Фар-Рокуэе.

Так что там с этими компьютерами?

Вы правы: я считаю, что мы потеряли большую часть мастерства, перейдя с логарифмических линеек на карманные калькуляторы. В последние годы я поражался, видя, как студент набирает цифры на своем карманном калькуляторе, получает явно нелепый ответ и отстает его, заявляя, что «так посчитал калькулятор». Мысль о том, что при вводе могла быть допущена ошибка или что можно попытаться каким-то способом оценить правильность ответа, явно не приходит ему в голову. Печально.

Вы скажете, что ничего страшного, компьютеры все это изменили?

Как бы не так. Но оставим эту тему. Гораздо интереснее посмотреть, как последнее поколение мастеров логарифмической линейки стало первым поколением профессиональных компьютерщиков. Ведь за 10 лет до того, как карманные калькуляторы вытеснили логарифмическую линейку, в университетах появились компьютеры, и те из нас, кто сумел заглянуть в будущее, увидели там компьютер.

Итак, мы изучали FORTRAN на машине класса IBM 1620. Не стану утомлять вас боевыми воспоминаниями. Но мы многое узнали о том, как перевести ручные вычисления в вычисления, производимые на машине. А поскольку эта процедура была весьма обременительной, мы писали программы только тогда, когда задача того действительно стоила, например для многократно проводимых вычислений.

Даже для тех из нас, кто почувствовал себя в программировании как рыба в воде, работа на компьютере была сущей мукой. Оказалось, что эти машины ужасно привередливы к расстановке знаков пунктуации и к случайным вторжениям в шестую колонку. (Если что-то было пробито в шестой колонке перфокарты с командой FORTRAN, это означало, что данную команду нужно присоединить к предыдущей, т. е. это было «поле продолжения». Естественно, когда это происходило случайно, компилятор FORTRAN с негодованием плевался.) Нам постоянно казалось, что наши программы отвергаются машиной по совершенно дурацким причинам.

В действительности все эти неудобства были следствием пакетной обработки. Единственная ошибка прерывала весь процесс, а исправлять их приходилось последовательно, по одной. Если не было возможности в течение дня выполнить несколько «проходов», то отладка затягивалась на очень длительное время. В результате мы старались быть аккуратными, потому что, нравилось нам или нет, машина ничего не прощала. Мы также стали помещать в программы множество операторов печати для диагностики,

чтобы в критической ситуации можно было понять, когда, где и желательно почему произошла авария. Самое главное, мы стали первыми программистами «оборонительного стиля». С появлением интерактивных терминалов, а затем персональных компьютеров этот стиль стал менее распространен, но, поверьте, значение его не уменьшилось.

Мы также совершили открытие, которое вслед за нами будут повторять еще 40 лет: большинство задач университетского уровня являются «учебными» и не требуют слишком больших или тщательно сделанных программ. И по иронии судьбы мы не тратили много усилий на код, проверяющий правильность входных данных. Ведь компьютер – это инструмент для профессионалов, т. е. инженеров. Представить себе, что компьютером станут пользоваться «обычные граждане», а потому потребуются проверять вводимые данные, было так же сложно, как вообразить прогулку человека по лунной поверхности.

Получились ли из нас более умелые пользователи компьютеров благодаря тому, что мы владели логарифмической линейкой? Уверен, что да. Нас приучили подозрительно относиться к результатам любых вычислений, кто бы их ни выполнил – человек или зверь (компьютер считался зверем). Поэтому ряды результатов, аккуратно отпечатанные на бумаге с дырочками перфорации по обоим краям, не оказывали на нас гипнотического воздействия. Для нас не существовало GIGO.¹ Мы знали, что результат может быть ошибочным. Лишь после тщательной проверки мы могли согласиться, что на сей раз компьютер не наврал. Так же, как в случае карманного калькулятора, мы поражались, насколько часто люди слепо принимают результат только потому, что его напечатал «компьютер». Мы знали, что программу, давшую этот результат, написал человек, и мы знали, как легко сделать ошибку, когда пишешь программу. Мы слишком много их написали сами, и у нас был печальный опыт.

Наше вычислительное наследство

В шестидесятые годы существовало поколение инженеров, обученных пользоваться как логарифмическими линейками, так и компьютерами. Тот, кто учился раньше, никогда до конца не принял компьютеры, а тот, кто учился позднее, так не научился толком считать вручную. Но эта группа молодых инженеров в наиболее активный период обучения застала инстру-

¹ GIGO – акроним от «Garbage In, Garbage Out» (мусор на входе – мусор на выходе).

менты как предыдущего, так и последующего поколений в тот момент, когда различие между теми и другими было наиболее существенно. На этой основе выросла целая культура отношения к результатам инженерных расчетов. Среди ее догматов были следующие:

- Не принимать ответ только потому, что сам его вычислил. Подвергай сомнению все. Когда у нас появились компьютеры, из этого возник принцип «не верь входным данным».
- Разбивать сложные вычисления на части разумного размера и проверять все промежуточные результаты, чтобы избежать размножения ошибок.
- При ручных или машинных расчетах придумывать способ проверки, позволяющий обнаружить явно нелепый результат.

Это были не абстрактные идеи, а практические рецепты выживания. И они превратились в инстинкты, позволявшие выжить в холодном и безжалостном мире решения задач.

Моя теория состоит в том, что это золотое поколение инженеров стало лидером того колосса, каким является сегодня Силиконовая Долина, и что приближение этого поколения к пенсионному возрасту представляет для нас опасность. Компьютеры стали больше, быстрее, мощнее. На них выполняются замечательные программы. Но кто сможет заметить, что они выдали неверный результат?

Резюме

Эта глава посвящена Эндрю Мараско (1916–2004), который не успел увидеть книгу напечатанной.

Ужасно трагично, когда элегантная теория подвергается атаке со стороны грубых фактов. В прошлом году мой друг Боб Маршал прислал мне статью из «*Invention and Technology*», озаглавленную «Как Америка не стала соперничать со Спутником». В этой статье¹ Т. А. Геппенгеймер (Т. А. Heppenheimer) утверждает: «Мы могли запустить искусственный спутник на орбиту Земли за год до Советов, но умышленно не стали этого делать. Дав им возможность осуществить пропагандистский триумф, мы обеспечили их конечное поражение в холодной войне». Интересно, не было ли это поражение отчасти обусловлено призывом целого поколения ученых и инже-

¹ Heppenheimer, T. A. «How America Chose Not to Beat Sputnik into Space». *American Heritage of Invention and Technology*, Winter 2004 Volume 19, Number 3, p. 44.

неров и не явилось ли одним из тех «неожиданных последствий», которое в данном случае оказалось нам на руку.

Какое отношение имеет весь этот экскурс в инженерное дело и вычисления к разработке ПО и руководству этим процессом? Он имеет отношение к тому, как решать задачи. Нас учили делать оценку, затем проводить вычисления, затем сравнивать результаты. Что делать, если результаты вычислений сильно противоречат вашим первоначальным оценкам? Надо рассмотреть три сценария:

1. Проверить, не было ли ошибки в вычислениях. Все хорошо, но вы напутали в числах.
2. Вычисления проведены правильно, но есть какая-то ошибка в модели. Например, вы предположили, что зависимость линейная, а она должна быть квадратичной. Явление вы выбрали правильно, но смоделировали его неверно.
3. Модель корректна, но выбрана неправильно. Имеется в виду, что действуют какие-то другие факторы, оказывающие влияние на вашу модель. Ваша модель правильно предсказывает результаты, но их искажает какой-то другой эффект, действующий в том же или в противоположном направлении, а вы этот эффект просмотрели.

Такой подход оказывается достаточно общим и потому применим к программному обеспечению. Например, я часто обнаруживал, что если отлаживать программу, дающую странный результат, именно в таком порядке, то удастся обнаружить истину. Прежде чем подвергать сомнению базовые предположения, проверьте арифметику (или правильные ли вводятся данные и вовремя ли, если это программа). Затем проверьте модель – в данном случае алгоритм. Если по-прежнему не ясно, в чем дело, посмотрите, нет ли чего-то, чего вы не поняли. В результате может обнаружиться, что подход к программированию был ошибочным с самого начала.

Наша подготовка в области вычислений имела глубокие последствия и привела к склонности почти все подвергать сомнению. Как я уже говорил, мы были первыми программистами «оборонительного стиля». Впоследствии в качестве менеджеров мы были требовательны к себе и к своим сотрудникам в соблюдении хорошей инженерной практики. Это означало усиленное внимание к базовой архитектуре и таким полезным вещам, как периодический пересмотр проекта и частое тестирование, начиная с ранних этапов.

Мы не пропускали мелочей. Мы знали, что если результаты «почти сходятся», это не доказательство их верности. Иногда в финансовых отчетах вы обнаруживаете, что итоги лишь чуть-чуть расходятся, и соблазнительно объяснить это «ошибками округления». Но очень часто оказывается, что совершены две существенные ошибки разного знака, почти погасившие одна другую. Когда боги вычислений милостивы к нам, они подкидывают такие ключи. Не поленитесь воспользоваться ими и всегда помните золотое правило Ричарда Хемминга: «Цель вычислений – понимание, а не числа».¹

Наконец, благодаря нашей инженерной подготовке мы были знакомы со сложностью и масштабированием. Я всегда буду помнить свой курсовой проект на старшем курсе. С целой группой студентов мы должны были с чистого листа сделать проект нефтеперегонного завода. Мы работали день и ночь в течение нескольких недель. И вот что выяснилось: трудно координировать работу нескольких инженеров, одновременно работающих над разными частями проекта, потому что продукты на выходе участка, разрабатываемого одним, оказываются входными продуктами для участка, разрабатываемого другим. Все приходится менять одновременно! Чтобы решить эту проблему, мы тщательно разработали интерфейсы и смогли сделать их устойчивыми. Много лет спустя эта идея оказалась незаменимой при создании больших и сложных программных систем. И другие идеи из науки «построения больших объектов» были так же с пользой заимствованы и пересажены на почву программирования.

¹ Ричард У. Хемминг (Richard W. Hamming) был одним из величайших прикладных математиков XX столетия. Он сделал блестящую карьеру в Bell Laboratories, а потом долго преподавал в военно-морском колледже в Монтерее (Калифорния). Он много печатался, и его хорошо написанные книги оказали на меня большое влияние.

ГЛАВА ТРЕТЬЯ

Альпинизм



В начале 1970-х я жил и работал в Швейцарии и вступил в Швейцарский альпийский клуб и провел несколько замечательных летних отпусков, начав занятия альпинизмом с нуля и достигнув среднего класса мастерства. Восхождение в горы больше соответствовало моим вкусам, чем скалолазание; возможно потому, что восхождение – более разнообразный вид спорта, и, в конце концов, скалолазание – его подраздел. А может быть, потому, что для восхождения требуется скорее выносливость, чем отдельные атлетические усилия. Во всяком случае я выяснил, что для успеха в этом деле нужны как техническое мастерство, так и умение строить отношения с людьми и принимать здравые решения. Я обнаружил, что альпинистов много, а тех, кто способен руководить экспедицией мало. Поэтому мне было интересно выяснить, благодаря каким качествам становятся хорошими руководителями.

Все те, кому я старался подражать, достигли своего успеха тогда, когда уже явно прошли пик своей физической силы, но накопили большой запас рассудительности и умения общаться с людьми. Хотя формально они еще обладали достаточными силами, чтобы быть первыми в связке, но их руководящая роль обеспечивалась другими качествами.

Позднее, когда в мои обязанности вошла оценка качества менеджеров программных разработок, я обратил внимание на большое сходство между хорошими менеджерами и теми ведущими альпинистами, с которыми я сталкивался в горах. Из этого я сделал вывод, что оба вида деятельности имеют между собой много общего. Это привело меня к представлению

об альпинистской экспедиции как хорошей метафоре проекта разработки ПО. Более глубокая разработка этой метафоры заставила меня полюбить ее еще сильнее, хотя к любым метафорам следует относиться с осторожностью и не заходить слишком далеко.¹

О восхождении на высокие горы

Создание большой программной системы очень напоминает подъем на большую гору. В обоих случаях цель вполне ясна и требует координированных усилий команды высококвалифицированных специалистов, находящихся в обстоятельствах, которые они могут предвидеть и к которым могут подготовиться, но которые им не подвластны. В обоих случаях успех имеет вероятностный характер, поскольку существует риск. В последующих главах я подробнее остановлюсь на некоторых деталях, но пока буду считать, что проект – это целенаправленная деятельность команды людей, и уделю особое внимание некоторым ключевым элементам, которые неизменно проявляются в каждом проекте.

Рассмотрим некоторые детали подробнее.

Уяснение объема задач как первый шаг планирования

В обоих видах деятельности хорошо выполненное планирование увеличивает шансы на успех. Первым шагом хорошего планирования является уяснение объема задач. Если речь идет о восхождении на гору, на этом этапе оценивается ее высота, относительная сложность местности, особые погодные условия и т. д. Едва ли можно себе представить, что в ответ на вопрос о высоте горы, на которую хотят подняться альпинисты, они ответят: «Когда заберемся, тогда и будет ясно!» Точно так же команда разработчиков должна представлять себе объем работы, за которую они берутся: насколько велика эта гора? Сколько понадобится строк кода, драйверов специальных устройств, необычных интерфейсов пользователя, какие нужны характеристики производительности? Ни сама бригада, ни ее руководите-

¹ Следует отметить, что к метафоре альпинизма часто прибегал Джеймс Хайсмит (James Highsmith) в своей книге «Adaptive Software Development: A Collaborative Approach to Managing Complex Systems» (Адаптивная разработка программного обеспечения: подход к управлению сложными системами на основе сотрудничества, New York, Dorset House Publishing Company, 1999). Свои собственные идеи я развивал независимо на протяжении многих лет и не знал об исследовании Джеймса, пока мне не указал на него один из рецензентов.

ли никогда не смогут предвидеть все трудности, но самые заметные характеристики задачи должны быть определены и зафиксированы, чтобы учесть их при последующем планировании.

На втором этапе планирования надо рассмотреть рамки проекта и возможные препятствия для его успешного выполнения, а также определить численность участников команды и навыки, которыми они должны владеть. Мой опыт, полученный в той и другой областях, подсказывает, что команда должна быть как можно меньше. У всех участников должен быть хотя бы минимальный уровень подготовки, чтобы они не тянули вниз всю команду. Например, если включение доктора в состав экспедиции оправданно, то он должен быть хорошим альпинистом и не быть обузой для команды, направляющейся к вершине. В той и другой области те, кто способен выполнять несколько функций, представляют большую ценность, чем специалисты: в сложном положении гибкость оказывается чрезвычайно ценной для небольшой команды.

Отбор участников

Подбирая в команду возможных участников, необходимо оценивать кандидатов по нескольким направлениям. Ясно, что нужно привлечь необходимых специалистов, ведь если придется, например, переходить сложный ледник, покрытый трещинами, значит, нужен альпинист, умеющий проходить ледяные склоны. А если вы создадите программный продукт с функциями, выполняемыми в реальном времени, вам нужен эксперт по времени выполнения программы. Кроме того, как уже отмечалось, надо стремиться отбирать альпинистов или инженеров с широкой компетенцией; если сравнить с профессиональным спортом, это соответствует стратегии отбора наиболее сильных атлетов, а не игроков на определенные позиции. Наконец, надо оценить характер, т. е. не только профессиональное мастерство, но и то, как человек будет работать в составе команды, переносить стресс и препятствия. Важно, чтобы будущий член команды оказался хорошим альпинистом и напарником как в хорошую погоду, так и в ненастье. Поскольку при большинстве восхождений без ненастной погоды не обходится, можно даже сделать поведение в таких условиях главным критерием отбора.

Есть один фактор, который более всего способствует снижению риска, – это опыт. Чем больше вы найдете людей, которые уже забирались на горы такого типа, тем лучше для вас. Альпинисты, побывавшие на вершинах многих гор, в большей мере обладают здравым смыслом, исходя чисто из дарвинистского подхода! Это называют «алгоритмом выбора проводника

на Аляске»: если есть выбор, остановитесь на том проводнике, который дольше всех занимается своим делом. У старых моряков есть поговорка: «отличный моряк благодаря своему отличному здравому смыслу избегает таких ситуаций, в которых ему понадобится его отличное мастерство». То же можно сказать об альпинистах и менеджерах программных разработок. Лучше призвать на помощь смекалку, чтобы избежать трудностей, чем справиться с ними ценой героических усилий. Самым простым признаком наличия здравого смысла является опыт. Ищите тех, кто уже делал это раньше и успешно. При прочих равных берите тех, кто благополучно перелез восхождение на крутую гору, а не тех, кто никуда не поднимался.

Вы можете себе представить восхождение на Эверест без шерпов?

Организация команды

Определив предполагаемый состав альпинистов/инженеров в своей команде, руководитель должен подумать о том, как разбить их на подгруппы. В альпинизме это означает, что на каждую веревку надо выделить по два, три или четыре человека.

Нечего и говорить, что независимо от сложности трассы все альпинисты должны участвовать в восхождении только в связке. Самостоятельный подъем, если какой-то альпинист по глупости и решился бы на него, представляет опасность не только для него самого, но и для всей экспедиции. Остальным придется принимать чрезвычайные меры для его спасения, если он допустит хотя бы малейшую ошибку. Точно так же в крупных программных проектах волк-одиночка представляет собой опасность для всех. Каждый должен быть в связке, состоящей хотя бы из двух человек.

Если требуется, чтобы инженеры в команде работали группами по трое или больше человек, старайтесь правильно их составить. Всегда должен быть сильный лидер, а расстановка по профессиональным и личным качествам должна быть такова, чтобы ни одна из веревок не провисала. Главным принципом должно быть равновесие: все веревки должны уравновешиваться, и вся совокупность групп тоже должна быть сбалансирована. Если не удастся создать удовлетворительный набор групп из имеющихся кандидатов, надо выяснить причины и при необходимости найти еще людей или отказаться от тех, которые есть.

Всегда стремитесь сформировать как можно меньше групп как можно меньшей численности каждая. Четыре группы по три или четыре участника могут составить очень продуктивную команду.

Помните, что в разработке ПО так же, как в альпинизме, скрыто действует коварная пирамида: чем больше людей вы собираетесь вывести на вершину, тем больше их потребуется для организационного обеспечения всего проекта. Этот рост примерно экспоненциально связан с высотой горы.

Составление календарных планов

Другим аспектом планирования является составление графиков – календарных планов. Определив состав команды в целом, можно прикинуть, сколько понадобится провизии и снаряжения. В программировании это соответствует определению качественного и количественного состава аппаратных средств и программного обеспечения и сроков их поставки. Чтобы правильно определить необходимые ресурсы, надо составить некоторое представление о том, как будет проходить восхождение. При этом возможны три типа ошибок:

- Недостаток ресурсов
- Излишек ресурсов
- Неправильно выбранный *тип* ресурсов, а значит, подъем бесполезного груза

Если восхождение совершают 10 человек за четыре дня, то нужен один тип ресурсов, а если это 15 человек и две недели, то положение совершенно иное.

Чтобы решить эту задачу, необходимо знать следующее:

- Маршрут движения
- Места промежуточных привалов
- Примерное время выхода в место привала с заданным количеством людей

Этого достаточно, чтобы набросать график и подсчитать, какие ресурсы необходимы для выполнения задачи. Не требуется во всех подробностях представлять, как вы будете добираться до каждой промежуточной стоянки. И это правильно, поскольку план, включающий в себя такие подробности, сопряжен со значительным риском: о большинстве из них нельзя сколько-нибудь достоверно судить, пока вы действительно не вступите на гору. Даже если расписать все до последней мелочи, детали будут меняться по мере продвижения. Однако если не считать непредвиденных обстоятельств, вынуждающих изменить маршрут, места стоянок, или *контрольные точки* (*milestones*), сильно изменяться не должны.

Контрольные точки полезны по двум причинам: они позволяют, во-первых, получить общее представление о том, как решить задачу и какие ресурсы потребуются, и, во-вторых, отслеживать продвижение к цели. Если вам казалось, что до первого базового лагеря вы доберетесь за два дня, а у вас на это ушло шесть дней, стоит присесть и задуматься, чего можно ожидать в дальнейшем. То, что у команды по-прежнему замечательный боевой дух, а лагерь лучше всех, которые вы когда-либо видели, имеет мало значения, поскольку весь проект, по-видимому, потребует по крайней мере втрое больше времени, чем вы планировали.

Верстовые столбы или арийские колышки?

Итак, я сторонник приблизительного восходящего планирования при общем согласии команды относительно времени прохождения основных контрольных точек. И я считаю, что очень важно контролировать время фактического их прохождения. При этом контрольные точки не должны быть слишком близки или, наоборот, слишком разнесены. Если восхождение рассчитано на 1–2 дня, то контрольные точки обычно отстоят одна от другой на несколько часов. При восхождении в Гималаях важные контрольные точки должны, по моему мнению, отстоять на дни. Для программного проекта продолжительностью от 18 до 24 месяцев правильным кажется выбор контрольных точек с промежутком в 3–4 месяца. Это соответствует примерно шести или около того контрольным точкам на весь проект. Если избран итеративный подход к разработке, то в порядке вещей иметь около шести «контрольных точек», даже если итераций больше. Например, двухгодичный проект может иметь 10 итераций, но только шесть из них потребуют самого тщательного контроля руководства по своему завершению.

Ничего плохого нет в том, чтобы у каждой подгруппы были свои более мелкие задачи, если они сочтут это полезным. Каждый руководитель такой группы сам должен решать, как организовать связку, чтобы двигаться согласованно по времени с остальными.

Контроль и учет

Большинство опытных руководителей команд, с которыми я знаком, стараются постоянно делать заметки. Если это альпинист, то во время каждого короткого привала из кармана достаются блокнот и огрызок карандаша и делаются записи. Изучая эти заметки после восхождения, я обнару-

живаю, что информации в них не много, но она очень конкретна. Обычно регистрируются время прибытия, отклонение от плана и возникшие необычные условия. Иногда встречаются неформальные замечания относительно поведения отдельных лиц или групп. Во время планирования нового восхождения такие заметки об аналогичных горах помогают уточнить начальные прикидки. Если это программный проект, то подобные заметки помогают предвидеть, как будет осуществляться действительный проект.

Учет рисков

Планы проектов должны учитывать возможность возникновения непредвиденных обстоятельств. В альпинизме *форс-мажор* представлен двумя независимыми переменными: выбором маршрута и погодой.

Если предварительно выбранный маршрут оказывается слишком опасным, то избирается какой-то другой. Обычно это происходит, когда ледник или скала оказываются в ином состоянии, чем они были во время последнего прохождения этого маршрута. В хорошем плане естественные остановки при восхождении служат для оценки возможностей и выбора альтернативных маршрутов. Точно так же при разработке ПО непрерывно оценивается техническое направление, и в зависимости от результатов каждой итерации путь для следующей итерации слегка меняется. При этом цель – пик вершины – остается неизменной. Однако изменение условий может повлиять на наше представление о том, как лучше всего до нее добраться. Редко когда существует только один путь, и талант альпиниста проявляется в том, что он находит правильный маршрут в изменившихся условиях.

Погода – это совершенно особый фактор. Когда она резко меняется, все ваше предприятие меняется коренным образом. Теперь ваша задача не добраться до вершины, а выжить. Даже когда все согласны, что дальнейшее продвижение невозможно и правильным будет «пересидеть» бурю, возможна гибель членов экспедиции, если пища кончится прежде, чем установится хорошая погода. Поскольку вы ни в коей мере не можете управлять погодой, относиться к ней следует с крайней осторожностью: сила вашей команды может потерять всякое значение. В горах больше всего людей гибнет из-за самолюбия: неспособность вовремя повернуть назад может стать роковой.

В программировании аналогичная ситуация возникает, когда вы зависите от таких вещей, которыми не можете управлять непосредственно. Это могут быть новая непроверенная технология, расчет на чудеса, необходи-

мость нарушить известные законы физики, внутренние и внешние поставщики и подрядчики. Игнорирование погоды в таких областях иногда приводит к гибели – либо мгновенной, либо мучительной и долгой.

Решительность

Я еще не встречал хорошего альпиниста, который был бы нерешителен. Но мне известны альпинисты, путающие решительность с безрассудством. Между одним и другим есть разница. (Можно добавить, что мне встречались старые альпинисты и встречались самоуверенные альпинисты. Но я никогда не встречал старых самоуверенных альпинистов.)

Главное, что я усвоил, занимаясь альпинизмом, – это то, что времени мучиться над принятием решения нет. Это не значит, что можно позволить себе стрельбу «от бедра». Часто приходится рассматривать сложные альтернативы, ситуации, в которых одно неверное решение может означать успех или неудачу всего предприятия, если не выбор между жизнью и смертью. Иногда правильный выбор неочевиден.

Однако нельзя позволить себе впасть в парализованное состояние и продолжать откладывать принятие решения – синдром «паралича при анализе». Необходимо потратить какое-то время и рассмотреть варианты, собрать данные, а потом принять решение. Приняв решение, надо выполнять его и не терзаться сомнениями.

Здесь возникают довольно тонкие задачи взаимоотношений в команде. Важно добиться общего согласия относительно существенных решений. Однако достижение консенсуса само по себе может привести к параличу. В какой-то момент, если возникает тупик, руководитель обязан принять решение. Если команда подобрана правильно, она подчинится, понимая, что лучше такое решение, чем никакого вообще.

Однажды я торчал на скале, не решаясь выбрать один из двух маршрутов, каждый из которых был хуже другого. Мой партнер по восхождению некоторое время терпел это, а потом сказал: «Либо ты примешь какое-то решение, либо будешь сидеть, пока не замерзнешь до смерти». То же самое происходит при разработке ПО.

Общая цель

Когда восхождение организовано хорошо, ни у кого нет разногласий относительно общей цели. Обычно она состоит в достижении вершины. Все, что не способствует достижению цели, безжалостно отвергается: ни-

каких прогулок в сторону, собраний цветов, получасовых остановок для фотографирования и т.п. Можно, конечно, заранее договориться, что какие-то из этих действий входят в программу и потому разрешены, однако важно, чтобы не было расхождений в отношении главной цели и средств ее достижения.

Аналогично при разработке ПО необходимо сосредоточиться на цели, которой обычно является создание программной системы. Интересные боковые маршруты могут подорвать весь проект, особенно если какая-нибудь из групп попадет в расщелину. Точно так же не стремитесь получить очки за артистизм. В альпинизме их не дают. «Некрасивое» восхождение на вершину всегда превосходит эстетически приятное отступление. Это не мое личное мнение, так подсчитывает очки практически весь мир.

Смотреть дальше

Когда рассуждают о восхождении на гору, часто ошибочно сосредотачиваются на том, как достичь вершины, хотя эта цель не единственная. Достичь вершины, а потом возвратиться всей командой вниз – вот настоящая цель. (Аналогично задачей космической программы шестидесятых было не высадить человека на Луну, а высадить его и вернуть обратно живым.)

С некоторой натяжкой в программном деле можно считать аналогичной ошибкой сосредоточение всех усилий на разработке и поставке первой версии продукта. Таким образом вы «поднимете флаг» на вершине. (Иногда о победе возвещают еще раньше, например при выпуске бета-версии.) Я считаю, что морально равноценным подъему на вершину и благополучному спуску обратно является выпуск программного продукта, который вы в состоянии сопровождать и поддерживать. Это означает, что сам программный продукт должен устойчиво работать, документация должна быть полной и понятной, а бремя поддержки не должно отнимать все силы организации. Поэтому планирование программного проекта должно охватывать всю работу, а не только установку флага на вершине.

Конкуренция может быть причиной неразумного поведения

Два наших сына, Дэйвид и Марк, были еще маленькими, когда мы жили в Швейцарии. Позднее они отчасти разделили мое увлечение альпинизмом. Дэйв обратил мое внимание на то, что «благородство» альпинизма частично основано на романтическом представлении о том, как некая команда борется со стихией ради достижения благородной цели. Он также

заметил, что на практике этот идеал часто оскверняется конкуренцией между командами. К одной и той же цели могут стремиться сразу две или три группы, каждая из которых хочет первой «водрузить на вершине флаг». (Некоторым областям научных исследований присущ аналогичный разрыв между романтическим идеалом и реальным миром.) Конечно, при разработке ПО давление конкуренции еще сильнее, поскольку разрабатываемый продукт является оружием в конкурентной борьбе, которое немедленно требуется всей остальной организации. Эффекты этого давления могут быть катастрофическими: часто изменения в плане, совершаемые на лету в ответ на требования конкуренции, заставляют команду идти на слишком большие риски, ведущие к провалу или худшим последствиям.

Обычные причины провалов

Отметив сходство между альпинистской экспедицией и проектом разработки ПО, я иду далее и утверждаю, что частично причины провала программных проектов такие же, как у альпинистских экспедиций. О многих из этих проблем будет многократно говориться в последующих главах. Возьмем некоторые из роковых ошибок альпинистов и посмотрим, какие аналогии есть в разработке ПО:

- Попытка слишком быстро взойти на вершину.
Аналог: Нереалистичный график работ.
- Попытка взойти на вершину с явно недостаточными ресурсами.
Аналог: Недостаток хороших работников или инструментов.
- Чрезмерная численность команды; трудности материально-технического обеспечения и коммуникаций оказываются непосильными.
Аналог: Слишком много посредственных разработчиков по отношению к первоклассным разработчикам.
- Затягивание экспедиции; команды, слишком долго остающиеся в горах, теряют живость, энергию и устремленность; усталость берет верх. Кроме того, может просто не хватить ресурсов.
Аналог: Никогда не завершающиеся программные проекты; они длятся так долго, что техническое задание меняется, иногда многократно.
- Движение по неверному маршруту, несмотря на новую информацию.

Аналог: Игнорирование результатов первых итераций. Отсутствие корректировки планов в ходе проекта.

- Поражение в силу обстоятельств, не зависящих от альпинистов.

Аналог: Невыполнение обязательств поставщиком или субподрядчиком; провал в разработке важной компоненты, которая в действительности не была готова к производству, а требовала предварительных исследований.

- Отсутствие разумного плана, который всем понятен, не вызывает сомнений в успехе и полностью принят к исполнению.

Комментарий: Обычно это результат нисходящего планирования, директив, спущенных сверху.

- Неспособность действовать согласно плану в пределах допустимых отклонений.

Комментарий: Иногда оказывается суммарным эффектом многочисленных мелких неудач, а не одного крупного провала.

- Потеря находчивости в случае затруднений; преодоление неожиданных неприятностей не рассматривается (ошибочно) как неизбежный элемент всего предприятия.

Комментарий: Одинаково плохо и в конторе, и в горах.

- Отсутствие резервов для критических ситуаций.

Комментарий: Обычно в некоторой мере такой резерв обеспечивается опытными участниками: они знают, как действовать в случае неожиданностей.

Составляющие успеха

Аналогичным образом можно, вероятно, обнаружить родимые пятна успешных команд, рассмотрев другие стороны этой метафоры. Вот некоторые из них:

- Успешные команды успешно осуществляют планирование, а не одержимы им.

Комментарий: Небольшой, но толковый план всегда лучше, чем обилие деталей.

- Маленькие команды способны быстро перемещаться (нужно успеть забраться на гору и спуститься с нее прежде, чем матушка Природа передумает и решит, что нечего вам делать на вершине в этот день).

Аналог: Завершите работу прежде, чем изменятся технические требования.

- Талант оценивать поступающие данные в реальном времени и вносить в план должные изменения в должное время.

Аналог: Использовать итеративную разработку, собирать систему на ранних этапах и часто и корректировать план на основе этих данных.

- Правильное соотношение между первоклассными участниками и хорошими членами команды и руководителями; обычно здесь очень важно как можно шире делиться информацией.

Комментарий: Необходимо равновесие и общее умонстроение.

- Сверка с разумно подробным планом.

Комментарий: Необходимо обнаруживать наступление неприятностей и знать, какие принимать меры.

- Лидеры демонстрируют зрелость и здравомыслие.

Комментарий: Необходимо знать, когда натянуть вожжи, а когда их ослабить.

- Выносливость: следует понимать, что средняя мощность за длительный период гораздо важнее, чем кратковременная пиковая. Не удивительно, что успешными руководителями экспедиций становятся не раньше, чем после сорока лет.

Аналог: Интересно было бы узнать, какова статистика для руководителей программных проектов.

- Стойкость: способность приложить все силы в трудных обстоятельствах.

Аналог: Существенно, например, при поиске трудно обнаружимой ошибки в программе.

- Сосредоточенность: команда не теряет из виду четко обозначенную цель.

Комментарий: Все участники знают что, зачем, когда и где.

- Творческий подход: готовность к экспериментированию и способность получать искреннее удовольствие от своей работы.

Комментарий: Как и везде в жизни.

Человеческий фактор

В каком-то отношении это сводится к тому, что «люди решают все». Что касается программных проектов, да и всякого другого стоящего занятия, я бы перефразировал романиста Ирвина Стоуна: «Дайте мне мужчин (и женщин), достойных моих гор». Человеческий фактор настолько важен, что далее я посвящу ему целый раздел.

Резюме

Многие менеджеры программных разработок, с которыми я обсуждал метафору альпинизма, указывали, что я упустил важный момент, а именно *масштабирование* (*scaling*). Они имели в виду, что общие принципы сохраняются независимо от величины горы или размера проекта. Детали, конечно, различаются: восхождение за пару воскресных дней на четырехтысячник с командой из 10 человек отличается от похода на Эверест. С другой стороны, и та, и другая экспедиции закончатся провалом, если будут нарушены общие принципы. Для очень крупных программных проектов нужна несколько иная организация, однако возникает предположение: не станут ли они более удачными, если применить к ним тот же подход, что и к маленьким командам, но только команд будет несколько? В обоих случаях представляется интересным вопрос о соотношении между ростом издержек обмена информацией при большом количестве команд и тенденцией к провалу, неизбежно усиливающейся при чрезмерном росте численности команды. Во всяком случае эта альпинистская метафора стимулировала возникновение многих интересных дискуссий о подлинной природе проектов разработки ПО.

Я по-прежнему являюсь членом Швейцарского Альпийского клуба, но теперь мои альпинистские подвиги по большей части ограничиваются стараниями выбраться из особо глубоких песчаных ям на площадках для гольфа в разных частях света.

Следующая глава посвящена некоторым мыслям относительно общих принципов управления, сформулированным мною за годы работы.

ГЛАВА ЧЕТВЕРТАЯ

Менеджмент

О менеджменте было написано, вероятно, больше чепухи, чем о каком-либо другом предмете. Если все так хорошо разбираются в этом деле, то почему рядовые работники столь единодушно презирают его?

Это обстоятельство долгие годы вызывало мое недоумение. Я помню, что в молодости был подвержен известной болезни считать всех своих начальников идиотами. Несколько позднее я понял, что это было просто разновидностью тех чувств, которые Марк Твен испытывал в 15-летнем возрасте к своему отцу. Я начал понимать, что эта работа не так проста, как мне вначале казалось. В некотором смысле мне повезло: когда мне впервые пришлось руководить людьми, я был несколько старше, чем обычно бывает новоиспеченный менеджер. Поэтому я, вероятно, делал меньше ошибок в начале своей менеджерской карьеры.

Тем не менее я полагаю, что и на мою долю досталось немало промахов, и некоторые из них были вопиющими, можете мне поверить. Рассматривая их теперь в ретроспективе, я нахожу кое-что утешительное для себя. Во-первых, попадая впросак, я всегда испытывал боль – обычно по своей собственной вине. И это было полезно, поскольку ситуация закреплялась в памяти – примерно как у ребенка, коснувшегося раскаленной плиты. Мне приятно считать, что я редко повторял свои крупные ошибки, предпочитая разнообразие. Если серьезно, то я стал интересоваться тем, как ведут себя действительно хорошие менеджеры. Найти хорошие образцы для подражания трудно, но если это удастся, важно тщательно проанализировать, благодаря чему они достигли успеха. Я старался чаще находиться рядом с таки-

ми людьми, которые иногда (но не всегда) охотно передавали свой опыт, и пытался узнать от них как можно больше секретов. Я придерживался мнения, что жизнь слишком коротка, чтобы самому совершить все ошибки, и что гораздо эффективнее учиться на ошибках, сделанных другими.

Управление командами

На основании приобретенного тяжелым трудом опыта руководства и управления коллективами людей я постараюсь выделить несколько базовых рекомендаций для достижения успеха. Описываемый мною сплав стратегий в области руководства и управления пригоден для сфер разработки продукта и обслуживания. Если вы когда-либо занимали руководящую должность, то, возможно, обнаружите, что я формулирую вещи, которые вы сами открыли благодаря своему опыту и здравому смыслу.

1. Сосредоточьте внимание на сколачивании сильной команды, способной решать сложные задачи и создавать реальную дополнительную ценность для клиента

Ключевые слова здесь следующие: *сосредоточение внимания, команда, сложные задачи и клиент*. Концентрация необходима, иначе ваша энергия будет расходоваться нерационально.

А поскольку нужные вам результаты будет в конечном счете создавать именно ваша команда, вам необходимо сосредоточить внимание главным образом на формировании и поддержке этой команды.

Самое удачное определение команды я нашел у Катценбаха и Смита:¹

Команда – это небольшая группа людей с дополняющими друг друга профессиональными навыками, разделяющих общую цель, стремление к эффективности и подход к работе, за которые они испытывают взаимную ответственность.

Первая трудность состоит в том, чтобы найти людей с правильным сочетанием профессионального мастерства и личных качеств. Кроме того, надо держать команду в форме, для чего поставить перед ней достойную цель – справиться с решением трудных задач. Нет смысла формировать превосходную команду, чтобы расслабить ее, заняв чем-нибудь тривиальным.

¹ Jon R. Katzenbach (Джон Р. Катценбах) и Douglas K. Smith (Дуглас К. Смит) «The Wisdom of Teams» (New York: Harper Business, 1993).

В центре задач должны быть проблемы клиента. Старайтесь не ставить перед командой внутренние автономные исследовательские задачи. Если все время помнить об интересах клиента, то гораздо выше шансы выбрать реальную практическую цель. Еще важнее, чтобы ваш продукт создавал какую-то реальную ценность для клиента. Иногда для этого приходится выяснить, каковы действительные проблемы клиента – в противоположность его мнению об этом.

*2. Лидеры вдохновляют, менеджеры обеспечивают.
Чтобы быть одновременно хорошим лидером и хорошим менеджером, необходимо уметь передавать общую концепцию продукта и разбираться в деталях*

Между лидерством (leadership) и управлением (management) есть разница. В идеале каждый из нас воплощал бы лучшие качества выдающихся лидеров и эффективных менеджеров, избегая стандартных ошибок тех и других.

Лидеры часто обладают харизмой, но это для них не обязательно. Лидер, демонстрирующий спокойную решимость и крепкую выносливость, может оказывать такое же вдохновляющее влияние, как и тот, кто зажигает сердца. Что отличает лучших лидеров, так это способность передать свое чувство цели – концепцию – остальным членам команды, постоянно их воодушевляя: их лидерство поддерживается собственным примером. Это вдохновляет команду на подвиги.

Менеджерам, как и лидерам, тоже необходимо иметь представление обо всей картине (видение), в рамках которой действует проект. Но они должны также разбираться во всех деталях, посредством которых команда сможет реализовать это представление. Менеджеры должны обеспечивать движение вперед: планировать, согласовывать, следить за настроением и устранять препятствия. Такая работа может быть эффективной только при условии понимания ими деталей. Чем более технически сложной оказывается проблемная область, тем важнее это понимание.

Менеджеры и лидеры должны знать, чем занимается каждый из них, но помнить при этом, что у каждого своя специализация. Основная задача лидера – передать общее представление. Основная задача менеджера – разбираться в деталях и обеспечивать эффективную работу команды и ее продвижение вперед.

Не часто выдающиеся качества лидера и менеджера присутствуют в одном человеке. Если вам поручено сформировать команду, то найти человека, совмещающего в себе обе эти ипостаси, может быть затруднительно.

Лучше попробуйте понять, в каком качестве – лидера или менеджера – ваш основной кандидат проявит максимальную эффективность, а потом попытайтесь найти того, кто сможет дополнить его. Проанализировав собственные сильные и слабые стороны, вы упростите выбор партнера для себя, когда вы станете руководить или управлять следующим проектом.

3. Готовьтесь к проблемам и устраняйте их в зародыше

Тут нет ничего особенного. Большинство проблем кажутся мелкими, когда они относятся к отдаленному будущему или остались далеко позади, но в своей «временной окрестности» они угрожающе велики. Отчасти это эффект перспективы, но есть и другие более скрытые причины.

Простое наблюдение показывает, что если не обращать внимания на мелкие проблемы, то они вырастают до больших. Это справедливо, например, для чувства неудовлетворенности, испытываемого сотрудниками: если ничего с ним не делать, оно развивается и усугубляется. Лучше каждый день удалять налет с зубов щеткой, чем дать ему нарасти и разрушить зубы.

Некоторые действия, такие как приобретение капитального оборудования, по самой своей природе рассчитаны на длительный срок. Если заняться ими заранее, зарезервировав достаточное время, то можно ограничиться административными средствами. Выделить средства на капитальное оборудование, заказать его, включить в план, установить и т. д. Обычно при этом не бывает трудностей. А если попробовать обойтись без подготовки? Ох, и побегаєте вы, когда отсутствие оборудования застопорит работу. Мелкая трудность станет крупной.

Обычно в эту ловушку попадают две разновидности представителей породы менеджеров: «страусы» (увиливающие от проблем) и «ленивцы» (откладывающие все на потом). Страусы не поднимают голову из песка, чтобы посмотреть, какие препятствия уже есть и еще грозят в будущем, поэтому они постоянно пребывают в состоянии неприятного удивления. Ленивцы знают о проблемах, но откладывают принятие каких-либо мер. Проблемам, однако, не свойственно обижаться и уходить, даже если на них никто не обращает внимания.

Хорошее руководство и эффективное управление предполагают активный поиск возможных препятствий и устранение найденных. Неожданность удара не может служить оправданием: правильное руководство – это искусство разумного предвидения.

4. Не ленитесь внимательно выслушивать людей, но не принимайте их мнение слишком близко к сердцу

Не слушать людей – это большой грех. Если вы считаете, что слишком заняты для этого, значит, вы неверно расставили приоритеты. Необязательно выслушивать всех и каждого, необязательно и прислушиваться ко всем одинаково, но слушать необходимо.

Ученые знают, что, игнорируя данные, они подвергают себя опасности. Возможно, что собрав какие-то данные, вы положите их под сукно. Но данные надо получить. Причем, по возможности, из первых рук. Исходные необработанные данные всегда имеют ценность, даже если они избыточны, потому что позволяют выполнить перекрестную проверку.

Помните, однако, что нельзя слепо идти на поводу у тех, кто предоставляет вам данные. У вас могут возникнуть идеи, которые покажутся странными и не понравятся другим, и они сообщат вам об этом. Послушайте их, взвесьте все – и слова, и музыку – и решите сами. То, что говорят другие, не должно переубедить вас, если вы знаете, что следует делать. Вам платят не за популярность в народе, а за выполнение задания. Если вас будет слишком беспокоить чужое мнение, то вы уподобитесь флюгеру: станете менять направление при каждом изменении ветра.

Этот совет спорен, поскольку мы живем в такие времена и в такой культуре, когда предпочтение отдается консенсусу. Но решения, основанные на консенсусе, могут быть ошибочными либо представлять собой неудачные компромиссы, на которые команда идет в условиях жестких временных ограничений. Если ваша команда не может достичь твердого консенсуса и возникает паралич, вы как лидер обязаны принять решение и двигаться вперед. В таких случаях любое ваше решение вызовет недовольство некоторых или даже всех участников – по крайней мере на некоторое время. Вы обязаны разъяснить всем, что *непринятие решения* – худший из путей. Решение все равно придется принимать, а пока это не сделано, теряется ценное время. Лучшее, чего можно добиться в такой ситуации, – уверить всех участников, что «суд их выслушал». Они не обязаны соглашаться с решением, но должны с ним смириться. Это совершенно необходимо для успеха команды.

5. Сосредоточьтесь на фактах

По ряду причин мы часто игнорируем этот принцип, что обычно ведет к катастрофическим последствиям. Будьте реалистом. Считайтесь с реаль-

ностью, а не с тем, что вам желательно, что могло бы быть, что может возникнуть в будущем. Живите в настоящем времени и работайте с фактами.

Отделяйте факты от мнений. Кроме того, отделяйте факты от их последствий и смысла. Часто обо всем этом говорят одновременно или путают друг с другом.

Обсуждая, оценивая, критикуя и т. д. все, что касается продуктивности, концентрируйтесь на фактах, а не на личностях. Оценивайте данные, основываясь на их фактическом содержании, и абстрагируйтесь от источника получения. Сначала соберите факты, а суждение по ним выносите позднее.

Я пришел к выводу, что регистрация происходящего помогает мне сосредоточиться на фактах. При этом я иногда составляю списки, делаю записи в стандартном формате или просто записываю заметки на будущее. Во время таких действий становится вполне ясно, когда можно справедливо написать «является» вместо «по-видимому» или «похоже, что».

б. Старайтесь укрепить стабильность, играя роль не усилителя, а демпфирующего звена

Это важное качество.

Большинство каналов, по которым мы воспринимаем информацию, подвергаются воздействию «шума». Во всякой организации есть фабрика слухов, постоянно выдающая ложную информацию. Менеджеры и лидеры должны избегать усиления шума, чтобы не потерять за ним сигнал. Новая ситуация весьма редко бывает настолько же плохой или хорошей, насколько кажется на первый взгляд. Получите данные, уясните их, а затем примите соответствующее решение. Сдержанная реакция почти всегда лучше. Когда в организации вспыхивает кризис, помните, что «и это пройдет», и станьте примером для всех остальных. Чтобы успешно руководить, нужно не терять голову, когда все кругом безумствуют. Ваша задача – сглаживать пики и волны, восстанавливая равномерный ежедневный поток энергии.

В некоторых случаях приходится действовать быстро, и «неумеренная» реакция может даже оказаться предпочтительнее. Например, если кто-то переманит вашего ведущего сотрудника, сделав ему «предложение, от которого нельзя отказаться», ваша реакция должна быть незамедлительной, если вы надеетесь изменить ситуацию. Либо, если ваша команда добьется крупного успеха, можно не сдерживать себя и реагировать с открытым энтузиазмом. Такие исключительные ситуации легко определить, если они возникают.

*7. Никогда не объясняйте злонамеренностью то,
что можно объяснить некомпетентностью*

Если чьи-либо слова или поступки способны оказать на вас или вашу команду неблагоприятное воздействие, не спешите делать неправильные выводы: паранойя может доставить вам много неприятностей. Если этот поступок кажется вам неверным, поначалу считайте его ошибкой. Постарайтесь представить себе заблуждения, которые могли привести человека к такому действию. Попробуйте поставить себя на его место.

Только исключив все возможные «сценарии ошибки», можно рассмотреть возможность присутствия неблагоприятных мотивов. Почему? Подумайте о последствиях. Если вы предположите злой умысел и окажетесь неправы, то почти неизбежно наживете себе врага, а у врагов есть скверная привычка умножаться в числе. Глупо создавать их себе без всякой необходимости.

С другой стороны, если вы ошибочно предположите некомпетентность, то да, вас, возможно, обманули. Но вас обманут только один раз. Если вы дадите нарушителю возможность исправить свою «ошибку», ему неизбежно придется показать свое истинное лицо. В конечном счете вы заслужите доверие остальных членов своей команды, относясь с уважением к своему врагу.

Я полагаю также, что некомпетентность встречается гораздо чаще, чем злонамеренность. Может быть, это и наивно, но мне кажется, что статистика на моей стороне.

*8. Развивайте чувство юмора в противовес напряженному
стремлению: относитесь к работе серьезно,
а к себе – с легким сердцем*

Меня иногда называли человеком напряженным. У этого качества две стороны – светлая и темная.

Напряженность – это пламя, вспыхивающее от единственной искры. Она позволяет сосредоточиться и способствует передаче чувства решимости всей команде. Отказ сдаться даже во враждебных обстоятельствах имеет важное значение.

Но у этого чувства есть и обратная сторона. Оно противоречит античному идеалу умеренности во всем: нет такого понятия, как «умеренная напряженность». Если вы из тех людей, которые так просто не сдаются, будь-

те осторожны, не дайте избытку вашей напряженности повредить всей команде.

Хорошо, когда есть чувство юмора. Даже в момент кризиса иногда стоит отступить немного назад, чтобы увидеть абсурдность всего происходящего. Смейтесь. Подтрунивайте над самим собой. Признавайтесь в своих ошибках и гордитесь ими, даже если это неприятно. Надо полагать, вы уже заплатили за свою ошибку, и если вам удастся над ней посмеяться, вы получите хоть какую-то прибыль на вложенные средства.

Я здесь не имею в виду черный юмор, который едва ли лучше, чем отсутствие юмора вообще. Я говорю о реальной и здоровой оценке глупостей, которые свойственны человеческой природе и всегда сопутствуют работе в любой организации, попыткам создать что-то из ничего.

Команды прощают своим менеджерам множество грехов, но некомпетентность, лень, отсутствие награды за старание и отсутствие чувства юмора не входят в это число.

9. Не ограничивайте свою жизнь одной работой и прочитывайте 25 книжек в год

Перечитайте еще раз пункт 8. Работа – это часть нашей жизни. Иногда нас так захватывает то, чем мы занимаемся, что «жизнь» кажется лишь придатком работы. В такой ситуации нарушается равновесие, но, не поддерживая постоянно равновесия, нельзя быть хорошим менеджером.

Нельзя эффективно руководить (и даже просто выжить), если нет каких-то иных интересов помимо работы. Для меня это были семья, физика, гольф и ряд других увлечений. Ходите в кино, в театр, играйте в покер, танцуйте, войте на луну – делайте то, что вам больше нравится. Но помните, что алкоголь – это депрессант.

Горячие ванны и долгие прогулки также помогали мне. Я пришел к выводу, что для сохранения формы нужно заниматься какими-то физическими упражнениями, даже если единственная мышца, которая вам нужна в течение всего дня, – это ваш мозг. Вот почему чемпионы мира по шахматам обязательно занимаются физкультурой.

Если стресс становится действительно невыносимым, поговорите с теми, кто не имеет отношения к вашему проекту и вашей компании. Они помогут вам обрести перспективу. В рамках компании самыми доверенными вашими лицами должны быть ваш главный архитектор и ваш босс.

Систематическое чтение – это еще один вид деятельности, имеющей важное значение для эффективности и выживания. С возрастом мы часто склонны повторять методы, эффективные в прошлом, вместо того, чтобы учиться и испытывать новые. Наши знания все больше расширяются благодаря опыту, а не через обычные каналы. Надо читать не только периодику. Поставьте себе цель прочитывать 25 книг в год, или одну книгу каждые две недели. Необязательно, чтобы они все были новыми, и необязательно, чтобы они все были техническими. Если ваша работа требует переездов, то чтение – это прекрасный способ полезного проведения времени, которое иначе вы попусту теряли бы в аэропорту или в самолете.

10. Доверяйте своим инстинктам: если к чему-то не лежит душа, то это может быть неспроста

Легко впасть в чрезмерное увлечение анализом. Иногда мы до посинения манипулируем цифрами, забыв о том, что за идея подвигла нас собирать эти цифры. Затем по этим цифрам мы делаем вывод, оставляющий ощущение неудовлетворенности.

В таких ситуациях обескураживает неспособность сформулировать причины нашего дискомфорта. И тем не менее, опасаясь, что такая извращенная ситуация парализует наши действия, мы упорно продолжаем анализировать, а потом принимаем решение, даже если не чувствуем, что оно правильное.

В большинстве таких случаев мне приходилось раскаиваться в принятом решении. И вот мой совет: если вы действительно чувствуете внутреннее сопротивление, доверьтесь своим инстинктам. «Внутреннее ощущение» не появляется за одну ночь, это чувство основано на всем вашем суммарном прежнем опыте. Во всяком случае заставьте себя разобраться в причинах своей неудовлетворенности и тогда попытайтесь избавиться от нее.

Лично у меня большинство таких неверных решений было связано с приемом на работу. Никогда не принимайте человека, с которым вы не чувствуете себя комфортно, хотя иногда можно и рискнуть. Однако если риск кажется вам достаточно большим и вызывает тревогу, доверьтесь чувствам и откажите этому человеку. Ошибки в подборе сотрудников оказываются одними из самых дорогостоящих.

Я всегда считал, что лучше принять плохое решение, чем не принимать никакого. Однако когда важные решения противоречат подсудным ощущениям, будьте очень осторожны!

Подводя итоги

Если изложенные здесь 10 мыслей показались вам собранием случайных безделиц, то так оно и есть. Менеджмент и лидерство – все еще виды искусства, а не науки; к дисциплинам, добавляющим в свое название слово «наука», – вычислительным наукам, управленческим наукам, общественным наукам – следует относиться с известной настороженностью.¹ Мы находим полезные правила эмпирически: пробуем что-то, следим, как часто это приводит к успеху или неудаче в длинной серии попыток, и пытаемся найти закономерность. Эти 10 идей приносили мне пользу на протяжении долгого времени, и я рассказываю вам о них лишь для того, чтобы вы сами могли оценить их и применить. Ваши результаты могут быть иными, потому что все зависит от способа применения.

Резюме

Как было замечено, Моисей не спустился с горы с Десятью Заповедями. Когда я вижу (или создаю сам) список из 10 пунктов, меня охватывает беспокойство. Даже над Вудру Вильсоном немного поиздевались за его «Четырнадцать пунктов»: его критики указали, что Богу для всей его мудрости хватило десяти.

С другой стороны, если бы я назвал этот список «10 советов Джо», это несколько принизило бы его значение. Эти идеи раз за разом оказывались полезными для меня. Они выведены из наблюдения в течение 40 лет за тем, как работает ряд очень успешных менеджеров. И когда я вижу, что кто-то грубо нарушает одно из этих правил, я нимало не сомневаюсь, что последствия будут неблагоприятными.

Хорошо, посмотрит читатель на этот список и скажет: а какое отношение это имеет к *разработке программных проектов*? Отмечу два обстоятельства. Во-первых, программный проект – это проект; принципы хорошего управления нельзя отбросить только потому, что ваш проект программный. Во-вторых, на протяжении ряда лет я отмечал, что разработчики

¹ Этим наблюдением со мною поделился Уэйн Мерецки (Wayne Meretsky), бывший когда-то разработчиком в Rational.

программного обеспечения несколько более цинично относятся к своим менеджерам, чем инженеры в других областях. Это довольно легко объяснить: они работают в быстро изменяющейся обстановке и с пренебрежением относятся к менеджеру, если тот хотя бы на полшага отстает технически, что немудрено. Единственный способ компенсировать этот понятный недостаток состоит в том, чтобы проявить исключительное мастерство управления. Потому что если разработчики решат, что с точки зрения вашего технического образования вы старая галоша *и* что к тому же вы никудышный менеджер, то едва ли вы сможете эффективно руководить ими.

ЧАСТЬ ВТОРАЯ

Особенности разработки ПО



Обратимся к тем разделам нашей науки, для которых одних лишь общих правил менеджмента недостаточно.

В главе 5 «Самое важное» я говорю о самом важном для успеха программного проекта понятии – об итеративной разработке. Оказывается, главнейший фактор своевременной поставки качественного программного продукта следующий: может ли команда эффективно применять итеративную разработку?

В главе 6 «Моделирование» обсуждается один из новых инструментов, позволивших поднять уровень абстракции и лучше объяснить суть ПО. Недостаток этого направления – некоторая недооценка написания кода, чем программисты заняты большую часть своего времени.

Поэтому в главе 7 «Кодирование» я кратко рассказываю о языках программирования и том, как разработчикам и менеджерам приспособиться к появлению *«идеального языка»*, регулярно происходящему примерно раз в 10 лет.

Наконец, в главе 8 «За дверь его!» я отмечаю, что успеху продукта должна предшествовать его поставка, а те продукты, которые разрабатываются вечно, редко оказываются прибыльными. Но оказывается, что «избавиться» от продукта не так просто, как может показаться. Я даю несколько советов тем, кто еще не освоил эту «досадную», но важную часть нашего дела.

ГЛАВА ПЯТАЯ

Самое важное

В этой главе вам предстоят два знакомства. Первое – с понятием итеративной разработки, а второе – с персонажем по имени Роско Леруа.

Итеративная разработка

Размышления о том, почему программные проекты заканчиваются успехом или провалом, многое помогли понять тем, кто занят в индустрии ПО. Из всех концепций, появившихся за последние 20 лет, итеративная разработка далеко опережает по важности остальные. Помимо очевидного закона, гласящего, что нельзя разработать очень хорошую систему, имея посредственных работников, важнейшее методическое соображение касается способа разработки ПО. Я многократно замечал, что команды, практикующие итеративную разработку, чаще добиваются успеха, тогда как те, которые применяют другие методы, чаще подвержены неудачам. Из методов успешной работы, пропагандируемых Rational Software, самым важным, по моему мнению, является итеративная разработка.

Чем это вызвано?

Основатели метода итеративной разработки отказались от чрезмерно жесткого водопадного (последовательного) подхода (waterfall approach), который доминировал в крупных проектах восьмидесятих годов. В его основе лежат снижение риска, инкрементное построение и оценка сделанной работы по реально работающему коду, а не по документам. Подробнее

об итеративной разработке можно прочесть у Ройса¹ или Кролла и Крухтена.²

В данной главе я рассказываю о некоторых технических причинах, по которым итеративная разработка постоянно доказывает свое превосходство. Но еще важнее, что здесь обсуждаются доводы в ее пользу, относящиеся к управлению. Это должно устранить разрыв между менеджером программной разработки и его менеджером, и вот что я имею здесь в виду. Итеративная разработка возникла в программных проектах и является их уникальной особенностью. Многим менеджерам общего профиля привычнее иметь дело с последовательным подходом, более близким к классическим методам управления проектами, с которыми они знакомы. Поэтому для данной главы возможны два способа употребления:

- Менеджер программной разработки может дать ее почитать своему менеджеру, которому станет легче понять, как осуществляется управление данным проектом.
- Главный менеджер может дать почитать ее своему менеджеру, руководящему программной разработкой, если тот не применяет итеративную разработку. При этом возникает вопрос, почему он так поступает.

В обоих случаях эта глава может послужить стартовой площадкой для разговора о том, как в данном проекте будут происходить управление, контроль и доклад руководству. Все это полезно обсудить раньше, чем начнется осуществление проекта, поскольку легче будет установить правила взаимоотношений между менеджером программных разработок и его менеджером. Это необходимое действие наряду с утверждением бюджета и подбором команды.

Роско Леруа

Первое, что вы должны усвоить: не стоит смеяться над именем Роско. «Оно показывает, что у моих родителей было чувство юмора. Кроме того, – добавляет он всегда, – могло быть и хуже, например, назови они меня Леруа Леруа».

¹ Royce, Walker, *Software Project Management: A Unified Framework* (Reading, Massachusetts: Addison Wesley, 1998).

² Kroll, Per and Philippe Kruchten, *The Rational Unified Process Made Kruchten, Philippe Easy: A Practitioner's Guide to the RUP* (Boston, Massachusetts: Addison Wesley, 2003).

Роско – старый боевой товарищ моего отца. По словам Роско, они оба служили под командованием Паттона, хотя я подозреваю, что Роско был непосредственным начальником отца, а Паттон находился где-то значительно выше. Роско один из тех старых морских волков, у которых есть свое мнение практически по любому вопросу. Неприятно то, что обычно он оказывается прав. Конечно, когда он неправ, это приводит к плачевным результатам, потому что он держится за свои убеждения до самого конца. Однажды я видел, как Роско выложил в покере все свои фишки на середину стола, будучи уверен, что его карты лучше, чем у остальных. Результаты были примечательны, но не совсем в том смысле, в каком хотелось бы Роско.

Его житейская мудрость проста и понятна. Она выражается в форме освященного временем совета: не лезь глубоко в теорию, и все будет в порядке. Поступать, как Роско, – это значит положиться на веру, что всегда полезно, когда настает решающий момент. Рассказывая, как он и там-то был, и тем-то занимался, он похвастается, что кончил всего восемь классов, но у меня есть все основания полагать, что где-то в тридцатых годах он окончил прекрасную школу. Отец объяснил мне это расхождение тем, что Роско просто не чувствует необходимости хвастать своим образованием.

Роско служил в армии, тушил пожары на нефтяных скважинах с Boots & Coots и работал горным инженером. Такой опыт сделал из него прагматика, но также научил действовать в жестких условиях. Если вы пригласите его на обед, то грязь из-под ногтей он вычистит, но не рассчитывайте, что он выберет правильную вилку.

Роско не прибегает к сложной математике, ему хватает арифметики. Обычно его ответы выражаются целыми числами. Если поинтересоваться у него, почему это так, он ответит примерно так: «Ну, в конечном счете все сводится к тому, сколько понадобится динамитных шашек и сколько человек для расчистки обломков. Но я же не буду распиливать шашки пополам и дробное количество людей тоже не стану нанимать на работу!». После такой формулировки ясно, что с высшей математикой покончено.

У Роско хорошая подготовка, твердые ценности и тьма опыта. Менеджер общего профиля из него замечательный. С точки зрения нашего повествования единственный его недостаток – это слабое представление о программировании.

Будем считать это не «изъяном», а характерной особенностью. Несколько лет назад, когда закрылась его шахта, Роско решил попробовать свои силы в разработке программ. Я взял на себя труд просветить его в этом деле, но, разумеется, он изрядное время занимался просвещением меня.

Оставляем последовательный подход

Как-то раз мы с Роско шли по его двору, собираясь поиграть в «подковки». Я сказал, что не играл несколько лет, но уверен, что обыграю его. «ОК, посмотрим, сынок, кто кого побьет, – сказал Роско. – Но прежде чем учить тебя, как правильно бросать подкову, я покажу, почему последовательная разработка обречена на провал».

Я невольно улыбнулся. Примерно за полгода до этого о разработке программ хотел поговорить я. Теперь уже Роско поднимал эту тему. Я не устаю искать новые идеи и очень заинтересовался, что же он мне расскажет.

Подобрав себе пару подков, я вдруг почувствовал, что эти крупные шутовчины в форме буквы «омега» оказались тяжелее, чем мне представлялось. «Ну, давай, – скомандовал Роско. – Вот столб. Посмотрим, попадешь ли ты с первого броска».

Естественно, после своей похвалы я чувствовал себя несколько неловко. Я сделал несколько пробных взмахов, принаравливаясь к подкове, и бросил ее. Конечно, на столб подкова не попала, даже намек на это не было. Она просто отскочила и приземлилась метрах в 5 от цели.

– Так нечестно! – возмутился я. – Ты должен был дать мне несколько пробных бросков.

Для тех, кто не знаком с этой игрой, надо сказать, что обычно перед началом соревнований делают несколько тренировочных бросков. То же бывает в любом деле. Необходимо приспособиться к своему снаряжению, которым в данном случае была моя рука, давно не практиковавшаяся. «Что ж, – сказал Роско, – это вряд ли помогло бы тебе, потому что есть три причины, по которым ты проиграл. Во-первых, как ты видишь, цель оказалась не там, где ее ожидала твоя рука, из-за некоторых ошибок калибровки.¹ Во-вторых, исполнение никогда не бывает совершенным; никто не бывает абсолютно постоянным – подкова не всегда летит туда, куда мы целимся. И, в-третьих, в промежутке между тем, как ты послал подкову и моментом ее падения на землю столбик переместился!»

¹ Оказывается, было две очень веских причины, по которым моя «калибровка» оказалась неверной. Подковы Роско были немного тяжелее стандартных, а расстояние между кончиками столбиков было немного больше нормы. Отклонения были невелики, около 10 процентов, и не сразу бросались в глаза, но из-за них цель оказалась «не там, где я ее ожидал». Вот почему так важны тренировочные броски: «компьютер», состоящий из мозга и тела, автоматически подстраивается во время пробных бросков.

– Чушь, – сказал я, – никто столбики не двигал.

– Отчасти верно, но после того, как ты бросил подкову, налетел порыв бокового ветра со скоростью 40 миль в час, который ты не учел. Эффект был таким же, как если бы кто-то переместил столбик в противоположном направлении, – ответил Роско. – Кроме того, угловое отклонение нужно помножить на расстояние, которое предстоит пролететь подкове, поэтому неудивительно, что ты промазал.

– Чем хорошо управление проектами – не обязательно каждый раз точно попадать, – сказал я.

– Это верно, – ответил Роско. – Потому и говорят, что «чуть-чуть не попал» засчитывается только тогда, когда бросаешь подкову или гранату. «Достаточной близости» часто оказывается достаточно при управлении проектом. Действительно плохо бывает тогда, когда промах оказывается большим, особенно, если твоя «подкова» летела долго. А при последовательном подходе такое происходит сплошь и рядом, потому что бросок у тебя всего один. При этом обычно, как и в «подковках», вмешиваются три важных фактора:

- Цель оказывается не там, где ты рассчитывал.
- Во время работы люди совершают ошибки.
- Вдобавок ко всему цель движется.

Это выглядело разумным. Соглашаться, что попытка будет единственной, видимо, глупо. Но я решил подразнить Роско, выступив с совершенно противоположных позиций.

Другая крайность

– Хорошо, я тебя понял. Давай попробуем забыть вообще о планировании и будем просто непрерывно уточнять свое местонахождение и направление дальнейшего движения, – мило улыбнулся я.

– Не шути, сынок, – ответил Роско. – Мы оба знаем, что анархия тоже неэффективна. Она просто влечет бессмысленное реагирование на мало-значительные события. По-моему, у вас, физиков, это называется броуновским движением. Бурная активность и слабый прогресс. Главная проблема вот в чем: любой проект должен осуществляться в виде последовательности этапов. И самое важное – это решить, насколько *продолжительным* лучше всего сделать каждый этап и сколько таких этапов должно быть.

Ясно было, что он готов к большому выступлению, и я хотел послушать его, поэтому я сказал: «ОК, Роско, то, к чему ты ведешь, куда интереснее, чем „подковки“, да и рука у меня побаливает. Сделаем перерыв».

Роско милостиво согласился.

– Да, выпьем по чашечке кофе, и я расскажу тебе о «коротких векторах», – сказал Роско. – А пока мы будем разговаривать, я сделаю для тебя несколько рисунков.

Итак, мы побрели к дому, налили себе кофе (кофеварка работает у Роско весь день) и перешли на заднюю террасу.

Первый рисунок Роско

Как только мы уселись в пару старых кресел-качалок, Роско вытащил из кармана огрызок карандаша и изобразил на бумажной салфетке то, что вы можете увидеть на рис. 5.1.

– Из геометрии нам известно, что в идеале кратчайшим путем между началом и концом проекта («целью») служит пунктирная линия, которую я только что провел. Те, кто придерживается последовательного способа, тешат себя иллюзией, что могут пройти по этому пути, но мы только что видели три причины, по которым это невозможно. Действительно хоро-

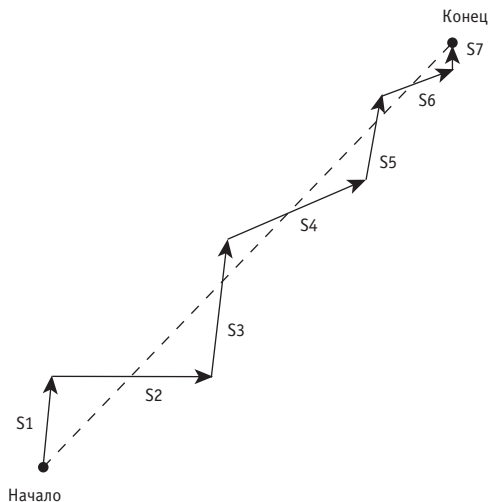


Рис. 5.1. Первый рисунок Роско – короткие векторы

ший менеджер проекта идет другим путем, типа того, который я нарисовал: из S1 в S2, оттуда в S3 и т. д. – делая ряд маленьких шагов, которые приводят к конечному результату. Каждый из этих шагов я называю *вектором* – так говорят на своем жаргоне авиадиспетчеры. У вас, программистов, каждый такой шаг называется *итерацией*.

– Напоминает парусник, лавирующий против ветра, – заметил я.

– Весьма, но останемся пока на позициях геометрии. Я хочу сравнить этот маршрут с тем, которым пошел бы менеджер проекта, если бы собрался обойтись всего двумя или тремя итерациями. – Роско глотнул кофе и взял другую салфетку.

Второй рисунок Роско

– Вот маршрут, который выбрал бы такой менеджер, – сказал он, нарисовав рис. 5.2.

– Обрати внимание, что здесь мы опять сделали упрощающее предположение, что цель не перемещается, хотя мы знаем, что это не так, – добавил он.

Я сразу заметил, что L1 идет в том же неверном направлении, что и S1 на предыдущем чертеже, и обратил на это внимание Роско. «Разумеется,

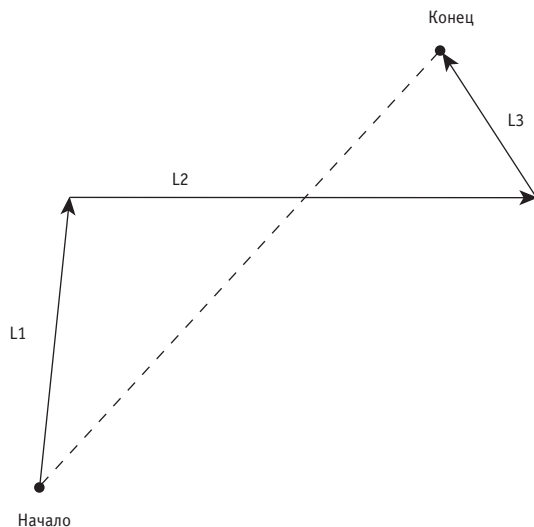


Рис. 5.2. Второй рисунок Роско – длинные векторы

они оба плохо начинают из-за ошибок прицеливания и исполнения. Но обрати внимание, что м-р Длинный Вектор остается на ложном пути гораздо дольше».

Кроме того, в первой точке, где производилась коррекция, оба менеджера – м-р Короткий Вектор и м-р Длинный Вектор – одинаково неправильно угадали направление дальнейшего движения: L2 и S2 направлены примерно одинаково. И снова м-р Длинный Вектор слишком долго упорствовал, прежде чем поменять направление.

«Теперь давай измерим длину всех трех маршрутов, – сказал Роско. – Пунктирная линия оказалась длиной 10 дюймов. Так... L1–L2–L3 протянулась примерно на 15 дюймов. А S1–S2–S3–S4–S5–S6–S7 – примерно на 12. Итак, м-р Короткий Вектор на 20% длиннее оптимальной величины, а м-р Длинный Вектор – на 50% длиннее».

Минуточку!

У меня возникло подозрение. «Постой, Роско! – вскричал я. – Я могу нарисовать для м-ра Длинный Вектор такой путь, который будет не длиннее, чем у м-ра Короткий Вектор».

– Верно, – ответил Роско. – С геометрической точки зрения совсем не обязательно, чтобы у м-ра Короткий Вектор всегда оказывался более короткий маршрут. Если мистеру Длинному повезет, то его результат окажется лучше. Но на практике все не так. Статистически усредняя многочисленные проекты того и другого, мы видим, что проекты м-ра Короткий Вектор оказываются эффективнее. Причина в том, что короткие векторы позволяют делать ряд других вещей, обычно невозможных при длинных векторах. Через минуту мы поговорим об этом. Ты прав, что нет геометрического доказательства данного обстоятельства, этот факт эмпирический, и все же геометрическая аналогия во многом склоняет в его пользу.

Я уже готов был принять рассуждения Роско, когда он вернулся к прежней точке зрения: «Ни один из этих рисунков не учитывает третьего фактора: вы стремитесь к цели, которая не стоит на месте. В отличие от столбика для набрасывания подков, в реальном мире цели всегда движутся. Это означает, что м-р Длинный Вектор рискует гораздо больше, чем м-р Короткий Вектор, потому что последний прицеливается чаще. Когда цель смещается, он меняет направление, а потому меньше времени тратит на движение по неверному пути».

Укорачиваем векторы

«Итак, – сказал Роско, – продолжим наш анализ. По нашей геометрической аналогии, если время и ресурсы пропорциональны длине пути, то м-ру Короткому Вектору потребуется их только на 20% больше, чем минимально нужно для успеха, тогда как м-ру Длинному Вектору нужно на 50% больше минимума. Поэтому у м-ра Короткого Вектора гораздо больше допустимая ошибка.

Можно взглянуть на это иначе, если посчитать объем отходов и переработки. Из 12 единиц, использованных м-ром Коротким Вектором, 10 нужны для конечного результата, а 2 потрачены напрасно, поэтому в отходы и переработку у него пошло около 16% всех затрат. У м-ра Длинного Вектора впустую потрачены 5 единиц из 15, поэтому у него отходы и переработка составляют 33%. Если пользоваться такой метрикой эффективности, то м-р Длинный Вектор действует вдвое хуже, чем Короткий».

Слова Роско были весьма убедительны. «Хорошо, Роско, – сдался я, – похоже, что короткие векторы представляют собой хороший универсальный принцип управления проектами. Но почему эта идея еще важнее в программных проектах?»

«С удовольствием объясню тебе, сынок, – сказал Роско торжествующе. – Но не принесешь ли ты нам сначала еще по чашечке кофе?» Затем он закурил свою любимую «стоджи».¹

Применение к разработке ПО

Когда я вернулся с кофе, Роско сделал глубокую затяжку и продолжил свои рассуждения. «Я считаю, – сказал он, – что для программных проектов обычно характерны следующие черты:

- Ресурсы очень ограничены; допустимая погрешность всегда очень мала.

¹ Министерство здравоохранения требует, чтобы я напомнил вам о вреде курения. Роско может прожить до ста лет, но если вы курите, то можете на это не рассчитывать. Кстати, у слова стоджи («stogie») любопытная этимология. Оно происходит от названия города Конестога в Пенсильвании, где изготавливали дешевые тонкие цилиндрические сигары, популярные в 1840-х годах. Возницы крытых фургонов «конестога» (изготавливавшихся там же!), наверно, курили такие сигары. Может быть, и дед Леруа управлял таким фургоном, другое название которого было «шхуна прерий».

- Вначале очень трудно точно «прицелиться»; как и в «подковках», нужно подобрать замах и начальную скорость. Можно сказать и по-другому: обычно вначале технические требования малопонятны.
- Ошибки исполнения неизбежны, особенно в начале.
- Цель *всегда* смещается по ходу проекта.
- Слишком большая задержка на неверном пути обычно оказывается катастрофической: приходится выбрасывать работу, которая считалась «выполненной», и начинать все сначала.

Если принять во внимание все эти особенности, то итеративный подход с короткими векторами начинает выглядеть очень привлекательно».

Аргументация показалась убедительной. Ясно было, что ошибки, особенно на ранних итерациях, могут оказать очень большое влияние, если их не обнаружить и не исправить. Когда я сообщил об этом Роско, он многозначительно кивнул.

Обучение на практике и выбор коротких векторов

– Ты не так глуп, как может показаться, – заметил он. – Работа должна совершаться короткими векторами, чтобы вся команда знала, как идут дела, и проводила соответствующие изменения курса. Накопив достаточно знаний и перенастраивая соответственно свои механизмы, ты совершаешь менее значительные ошибки в «прицеливании». С каждой очередной итерацией ты можешь действовать успешнее, исходя из того, что стало известно, поэтому все меньше отклоняешься от выбранного пути. И каждый раз, соединяя новые знания с обнаружением перемещения цели, ты можешь лучше предсказывать движение в будущем. Можно даже наловчиться и немного «упреждать» цель. Следовательно, метод коротких векторов приводит к укорочению суммарного пути по двум причинам. Во-первых, меньше времени тратится на ошибочные направления в начале пути (принцип коротких векторов). Во-вторых, последующие векторы становятся ближе к оптимальным (принцип обучения). Хотя и бывают счастливые исключения, но статистически именно эти две особенности ответственны за более короткий суммарный путь, если сделать усреднение по большому числу проектов.

«Есть ли систематическая технология максимально эффективного обучения на ранних итерациях?» – подумал я вслух.

Программирование рисков

– Да, и она называется *программированием рисков*, – ответил Роско. – Программирование рисков занимается выявлением опасных мест проекта. Допустим, что мы применяем абсолютно новую технологию. Она может увести наш вектор в совершенно неверном направлении, поскольку если эта технология окажется непригодной, мы потратим на нее массу времени, а в итоге откажемся от нее и выберем другую. Посмотрим, как можно разобраться с ней на ранних итерациях.

Роско стал излагать.

– Определив, что применение этой новой технологии связано с риском, мы должны расположить ее соответственно величине опасности в перечне других рисков проекта. Допустим, что она займет в этом списке первое место. Что мы делаем дальше?

– Опыт разработки программного обеспечения подсказывает мне ответ, – произнес я. – Сначала нужно устранить или снизить в следующей итерации риск применения именно этой новой технологии. Но согласно твоему принципу коротких векторов нам следует ограничить сферу действия каждой итерации. И поэтому спроектировать часть системы так, чтобы «выдать» эту новую технологию как можно быстрее и как можно тщательнее. Мы должны принять решение, причем обоснованно и быстро. Итерации надо проектировать примерно так же, как научный эксперимент: что мы хотим установить? как мы это установим? как сделать это с минимальными побочными эффектами?

– Даже я не сказал бы лучше, – согласился со мной Роско. – А что ты ответишь тем, кто скажет, что это уже не итеративная разработка, а создание прототипа?

– Конечно, это несколько напоминает создание прототипа, – продолжал я. – Но есть одно важное отличие: мы должны проверить технологию в контексте того, как она будет действовать в итоге в окончательном продукте, поэтому нужно также учесть масштабируемость и производительность. Экспериментировать на этой стадии с игрушками опасно, поскольку может возникнуть ложное чувство безопасности и самоуспокоенности. Хорошо спланированные итерации приведут к построению части системы, которое подтвердит или отвергнет применимость этой технологии. В конце итерации должна быть возможность оценить работающую программу и ответить на простой вопрос: продолжать двигаться в этом направлении или нет. Если да, то созданное во время этой итерации *войдет*

в окончательный продукт, в отличие от прототипа, который будет выброшен.

Подытоживая, я добавил: «Предпочтительнее тот эксперимент, который может показать непригодность новой технологии, а не тот, который усыпит наши подозрения относительно нее. Иногда, уступая желанию выбрать определенный путь – в данном случае это применение новой технологии, – мы планируем такие итерации или эксперименты, которые успокоят нашу тревогу. В результате в будущем нас могут ожидать большие неприятности, если обнаружится, что мы принимали желаемое за действительное».

– Да, – сказал Роско, – есть добрый совет: проявить жесткость в начале, чтобы не случилась беда позднее.¹ Программирование риска дает нам цель для каждой итерации. Итерация может быть нацелена на проверку нескольких рисков, но всегда нужно разбираться с ними в порядке очередности. И помни, что чем раньше начинаешь работать над сложными проблемами, тем больше времени останется на их решение, – продолжал он. – В связи с этим я вспоминаю одну очень неприятную ловушку, в которую на моих глазах попадали многие проекты ПО.

Я был весь нетерпение.

А эту песню ты слышал?

– Иногда я слышу, как менеджеры программных проектов говорят: к этому риску мы обратимся позднее, а пока займемся простыми вещами, и у нас будет время подумать. Когда я слышу такое, у меня давление подымается выше крыши! – воскликнул Роско.

– Ну, Роско, я знаю, откуда идет такой способ мышления, – сказал я. – Мы же сами предлагаем его своим студентам для решения контрольных заданий в условиях дефицита времени. Мы всегда предупреждаем их, что лучше сначала сделать простые задания, «поместить деньги в банк», а оставшееся в конце время посвятить сложным задачам. Логика в том, чтобы получить побольше баллов за то, что ты знаешь, а не «оставлять деньги на столе» из-за того, что не хватило времени.

¹ Джон Уокер (John Walker) напоминает нам о важности отсеивания порочных идей на ранней стадии. Придумайте способ «опровержения» метода и посмотрите, как это у вас получится. Иногда поиск в литературе приносит успех. Как убедительно сформулировал Ф. Х. Вестхаймер (F. H. Westheimer), «месяц лабораторных экспериментов избавит вас от часа сидения в библиотеке».

Роско на время задумался.

– Пожалуй, в каких-то обстоятельствах такую стратегию можно оправдать. Но дело в том, что у программных проектов есть свои особенности. Ты не получишь никаких баллов за незавершенный проект, как можно получить их за частично выполненную контрольную. Реализуя проект, ты *обязан* решить сложные задачи. Поэтому ситуация не просто непохожа, она диаметрально противоположна.

Я согласился.

– Стало быть, Роско, ты утверждаешь, что надежда на то, что команда станет работать над трудными или рискованными задачами в фоновом режиме, – это лишь бесплодная мечтательность, и вот почему:

- Пока команда работает над «простыми вещами», она не станет размышлять о рисках, отложенных на будущее. Команде трудно сосредоточиться одновременно на нескольких проблемах, и, начав работать над простой задачей, она совершенно забудет об отложенной задаче, связанной с риском. С глаз долой – из сердца вон.
- Согласно закону Паркинсона «легкая задача» неизбежно разрастется вширь, чтобы занять собой весь график работ. При этом более трудные задачи вытесняются. Я не раз был свидетелем такого хода событий. Потратив на 20–30% больше времени, чем предполагалось, на легкие задачи – иногда из-за чрезмерного перфекционизма – команда уже начинает отставать от графика. Вот тут она и сталкивается с настоящими трудностями. При этом проблема была известна давно. Она же не исчезла оттого, что ее игнорировали.
- В конечном итоге у команды оказывается *меньше* времени для работы над сложной проблемой, а не *больше*.
- И вот худшее, что может произойти: иногда для решения сложной проблемы приходится выкинуть все, что вы сделали, занимаясь «легкой задачей», и начать все сначала.

И в заключение я подвел итоги.

– Мораль сей истории: определите приоритет своих рисков и начните с самого опасного. Если в результате не останется времени на решение «простых задач», значит, вы в любом случае были обречены на провал.

– Вот теперь, сынок, ты попал в точку, – сказал Роско и дополнил мое наблюдение еще одним убедительным своим. – Если бы я выбился из графика, то мне было бы гораздо легче пойти к боссу и попросить у него дополнительное время, *сумею я доказать ему, что моя команда работает*

по плану, который уже позволил исключить большинство рисков. Напротив, если бы мне пришлось признаться, что мы не только отстали от графика, но нам еще предстоит разобраться с некоторыми проблемами, связанными с высоким риском, то мои позиции оказались бы очень слабыми. Вполне возможно, что мне не дали бы дополнительного времени, а проект бы закрыли. И, вероятно, были бы правы.

Еще об обучении на практике

Со всем этим я был вполне согласен: короткие векторы, целевое определение рисков и первоочередное обращение к сложным задачам. Одно мне не давало покоя: в итеративной разработке мы обязаны не только попутно учиться, но и *применять* то, чему научились.

Я обратил внимание Роско на то, что нельзя планировать следующую итерацию, пока не применено то, чему мы научились на предыдущей. В итеративной разработке нет места «автопилоту»: нельзя спать, когда меняется курс. В известном смысле итеративная разработка не только поощряет применение полученных знаний, но просто требует этого.

– Сравни это с организациями, которые не применяют итеративную разработку, – сказал Роско. – Часто ли приходилось тебе слышать, как люди говорили: «Ну уж в следующем проекте я такой ошибки не сделаю» или «Придется запомнить это на будущее»? Так происходит потому, что их план проекта настолько жесткий, что сразу учесть в нем полученные уроки они не могут. У них есть «зафиксированный» план, изменить который нельзя или очень трудно. Это печально по двум причинам: во-первых, команда смирилась с тем, чтобы «погибнуть вместе с кораблем», поскольку считает, что «слишком поздно» менять план; во-вторых, этот урок скорее всего будет забыт, когда появится новый проект, и весьма вероятно, что те же ошибки будут повторены вновь.

Вена на виске у Роско забилась, и я понял, что близится крещендо. – Это и есть «планирование, вышедшее из-под контроля», – воскликнул он. – Это все равно, что путать карту с местностью. Или объяснять провал плохими инструментами. Если действующий план ведет к катастрофе, МЕНЯЙТЕ ЭТОТ ПЛАН!!! – Он откинулся в своем кресле.

Я вынужден был согласиться. Итеративная разработка требует немедленно применять уроки, извлеченные из *этого* проекта, ко всем последующим итерациям *того же проекта*. Как объяснит вам любой преподаватель, обучение наиболее продуктивно и полезно, если как можно скорее

применять полученные уроки. Это очень важное различие между итеративной и последовательной разработкой. Итеративная разработка требует «ковать железо, пока оно горячо», даже если это сопряжено с болью: если применить правильную технологию, пока боль еще свежа, урок окажется глубже и вернее.

Следствия для бизнеса

Роско хотел сделать еще одно замечание. Он снова потянулся за салфетками и вытащил новый «холст» для своего завершающего эскиза.

– Ты наверняка видел вот такие кривые, сравнивающие каскадную (последовательную) и итеративную разработку, – произнес он, вычерчивая рис. 5.3.

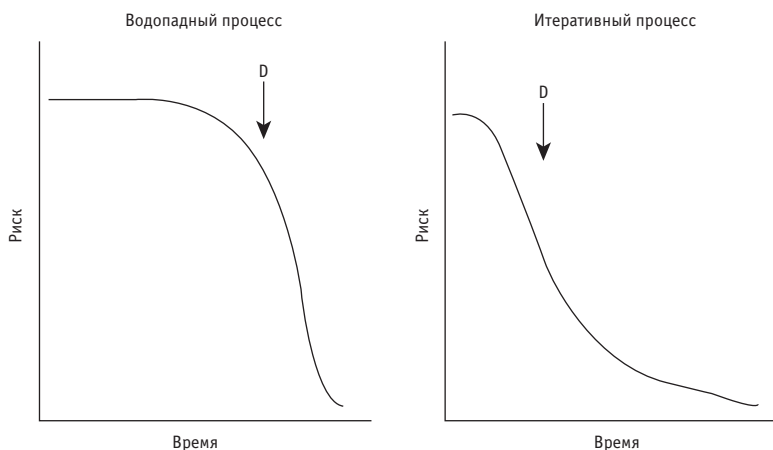


Рис. 5.3. Третий рисунок Роско – каскадная разработка против итеративной

– Основной смысл этих кривых в том, чтобы показать, что при каскадной разработке риски сохраняются гораздо дольше, поскольку, пока система не будет в достаточной мере собрана, мы не сможем узнать, как в действительности обстоят дела. При итеративной разработке, которая предполагает целевое обнаружение рисков, компоновку и сборку в достаточном объеме с первой же итерации, риски устраняются на ранних стадиях. На обоих рисунках точка «D» представляет собой момент принятия решения о продолжении или прекращении проекта. То есть если в какой-то мо-

мент кривая риска не пойдет резко вниз, мы можем решить, что продолжение невозможно; без сокращения риска проекта дальнейшие инвестиции в него невозможны или нецелесообразны.

– Конечно, Роско, – сказал я. – Я видел, как Дэйв Бернштейн¹ годами рисует эти кривые. Ничего нового тут нет. Ясно, что лучше принять решение раньше, чем позже. Поэтому последовательный подход уступает по качеству.

– Да, – возразил Роско, – но есть второе, менее известное обстоятельство, также свидетельствующее в пользу итеративной разработки. Я тебе сейчас его продемонстрирую.

Эффект численности персонала

Роско взял карандаш.

– Наложим на наши картинки кривую численности сотрудников. При последовательной разработке сотрудники принимаются на ранних стадиях, тогда как в итеративных проектах это происходит позже. Маленькие отборные команды, способные быстро продвигаться вперед, идеально подходят для выполнения основных работ на ранних этапах, т. е. на итерациях обследования и проработки.² Затем, на стадиях построения системы и передачи в эксплуатацию, добавляется много новых людей. Разница выглядит примерно так... – И он нарисовал новые кривые (рис. 5.4).

– Теперь допустим, что наш проект закрывается в точке D на обеих кривых. Ресурсы, израсходованные к этому моменту, показаны в виде заштрихованных областей под кривыми – площадями по кривой численности персонала от начальной точки до точки D, в которой происходит закрытие проекта. – Заштрихованные им области показаны на рис. 5.5.

– Сравни заштрихованные области. Итеративный сценарий предпочтительнее по двум причинам: решение о закрытии проекта принимается раньше, а затрат произведено гораздо меньше, поскольку проект начинался с малой численностью работников с намерением увеличить ее позднее.

¹ Дэйв Бернштейн (Dave Bernstein) более 20 лет управлял разработкой продуктов в Rational Software. Тот, кто может держаться в седле так долго, наверняка владеет какими-то секретами.

² Обычно проект делят на четыре фазы: обследования (Inception), проработки проекта (Elaboration), построения системы (Construction) и передачи в эксплуатацию (Transition). Каждая фаза состоит из одной или нескольких итераций. Мы вернемся к этому в главе 12.

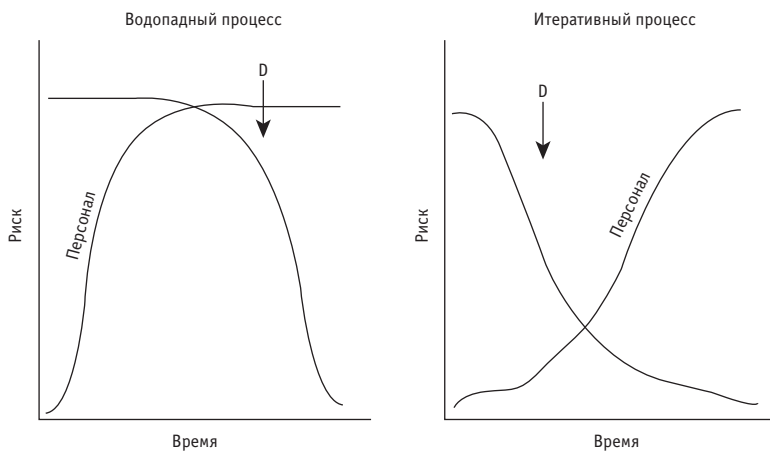


Рис. 5.4. Третий рисунок Роско – последовательный метод против итеративного, дополненный графиком численности персонала

Роско был прав, но мне хотелось бы отметить одну маленькую деталь. При итеративной разработке в начале проекта участвует гораздо меньше сотрудников, чем при последовательном методе, но обычно это более

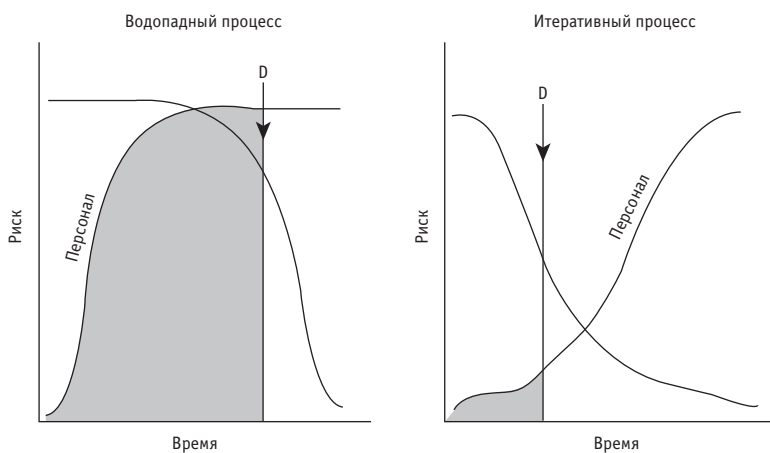


Рис. 5.5. Третий рисунок Роско – последовательный метод против итеративного, дополненный заштрихованной областью

опытные,¹ а потому высокооплачиваемые работники. Поэтому заштрихованные области отражают численность персонала, но необязательно затраты на оплату его труда. Но обычно данный эффект не столь существен, поскольку большее значение имеет разница в численности занятых.

Роско продолжил рассуждения.

– Отсюда можно сделать вывод, что появляется возможность начинать больше проектов с итеративной разработкой, потому что можно быстро прекращать их, неся относительно небольшие убытки. Хотя выгода от этого не столь высока, поскольку в большинстве организаций не так много действительно ценных работников, которых можно было бы использовать на ранних итерациях. Таких людей всегда не хватает.

Роско близился к завершению своего рассказа.

– Запомни также, что последовательный способ – это катастрофа, потому что с увеличением расходов на заработную плату (заштрихованной области) организационно и политически становится все труднее закрыть проект. Когда проект с итеративной разработкой закрывается на раннем этапе, трудоустроить нужно относительно небольшое число людей. Когда же закрывается последовательный проект на поздней стадии, это влечет перемещение многих людей.

Я понял его мысль. В конечном итоге, когда по какой-то причине приходится прекратить работу над проектом, с деловой точки зрения лучше сделать это раньше, прежде чем будут впустую истрачены существенные средства. Способ комплектации персоналом при итеративном подходе снижает эти расходы на начальных этапах до минимума.

Простой здравый смысл

Часто аргументация в пользу итеративной разработки бывает сосредоточена на «технологии» разработки ПО. Указывают на такие проблемы, как необходимость проведения исследований, неточность выдвигае-

¹ Отмеченный автором фактор усугубляется тем, что предполагается не только более «опытный» и «квалифицированный» персонал, но еще и люди, владеющие новыми и нетрадиционными программными техниками, которые именно и составляют «зоны риска». Дело не только в том, что такой персонал требует более высокой оплаты, но и в том, что его можно просто физически не найти в ближайшем окружении или достаточно трудно найти в требуемом числе. – *Примеч. науч. ред.*

мых требований, риски архитектуры, производительности, интеграции и т. д., которые оправдывают итеративность разработки.

Роско же продемонстрировал мне, что итеративная разработка оправдана с общей точки зрения управления проектами. Но в частном случае некоторые особенности, присущие разработке программных систем, еще более усиливают преимущества коротких векторов, первоочередного анализа рисков и применения полученных знаний. Таким образом, то, что полезно в общем случае, оказывается еще полезнее, когда в проекте создается ПО. Кроме того, разумная политика комплектования персоналом на ранних итерациях гарантирует, что, даже если проект придется закрыть, финансовые потери окажутся не слишком большими.

Как сказал Роско, погасив свою сигару и закинув ноги на перила террасы, «если это оправданно с точки зрения технологии и с точки зрения бизнеса, то это оправданно и для меня».

Резюме

Существует довольно широко распространенное мнение, что критиковать последовательный подход – это все равно что избивать мертвую (или не родившуюся) лошадь. Вопреки расхожему мнению последовательный подход, так сказать, жив и невредим и продолжает по сей день губить проекты. В разработке ПО от него отходят гораздо медленнее, чем хотелось бы.

В своей книге об управлении проектами Уокер Ройс указывает, что его отец, Уин Ройс, предлагал использовать последовательный подход гораздо более итеративным образом, чем это стали делать на практике. Очевидно, путаница существует даже в этом вопросе.

Вместо того чтобы обсуждать недостатки последовательного метода, гораздо более конструктивно, по моему мнению, сосредоточиться на полезных рекомендациях для итеративного:

- Уделите особое внимание архитектуре;
- Учитесь на ранних этапах, и это приведет к последующему росту эффективности;
- Последовательно и пошагово наращивайте систему с каждой итерацией;
- Судите о прогрессе по работающему коду, а не по документам;

- Тестируйте, соединяйте компоненты и выполняйте сборку на каждой итерации;
- Неуклонно снижайте риск, решая в первую очередь сложные задачи;
- Корректируйте план, тщательно изучая результаты предыдущей итерации;

Если у вас остались сомнения в отношении итеративной разработки, я могу только посоветовать вам найти менеджера, которому довелось применять оба подхода, и потолковать с ним. Дальнейшее изложение предполагает, что вы приняли на вооружение итеративный метод разработки ПО. С ним тесно связано и множество других концепций, которые я буду обсуждать. Например, я поведу речь о рабочей среде с «максимальным доверием», которая важна для эффективного применения итеративного метода. Еще одна близкая идея – это стабильная базовая «программная архитектура», благодаря которой в случае применения итеративного метода можно удерживать на правильном курсе сложный проект. Лучше всего описать эту архитектуру с помощью модели, поэтому следующую главу я посвящу моделированию.

ГЛАВА ШЕСТАЯ

Моделирование

Люди занимаются рисованием со времен появления наскальных изображений в Ласко, а может быть и дольше. Человек – животное *зрящее*,¹ и даже после того, как возник язык, мы продолжали вести исторические хроники и общаться друг с другом с помощью графики. Едва ли можно оспаривать, что наш «компьютер», объединивший зрение и мозг, служит мощным инструментом, который с чрезвычайной скоростью интегрирует невероятный объем информации, обеспечивая работу других механизмов познания. Это находит отражение даже в повседневной речи; поняв смысл чего-либо, мы часто восклицаем: «Теперь я вижу!».

Разработчики ПО в течение долгих лет пренебрегали зрительными способностями человека. Даже когда окружающие пытались описать создаваемые программы (и заставить программистов сделать это) с помощью элементарных рисунков, ответом было «давайте посмотрим код». И это было бы вполне приемлемо в разговоре двух разработчиков примерно одинакового уровня. Такой подход даже привел к появлению практики *ре-визии кода* (*code reviews*), что было неплохой идеей. Но как только вы поднимаетесь выше уровня подробной детализации, код становится едва ли не худшим возможным средством описания программы.

Дело в том, что при этом задействуется не тот уровень абстракции, который нужен. Попросту говоря, этот подход слишком детализирован. Как только вы пытаетесь описать взаимодействие различных частей кода внут-

¹ Поясню, что волка я бы отнес к преимущественно *обоняющим* животным.

ри подсистемы или взаимодействие подсистем между собой, вы оказываетесь беспомощны. Менеджеры слушают, изображают графически то, что, как им кажется, они услышали, показывают рисунки разработчикам и обычно получают в ответ: «Угу. Наверно. Может быть».

Здесь необходимо проявлять осторожность. Агитируя за графику, следует отметить ее ограниченность. Насколько графика позволяет передавать ценную информацию, настолько же она способна выразить различную ложную информацию. Речь идет не о детальных технических чертежах, содержащих неточности – последние быстро обнаруживаются. Подлинно опасными оказываются картинки «уровня менеджмента», какие часто встречаются в презентациях PowerPoint. Эти картинки претендуют на то, чтобы сообщить нам что-то, но на практике обычно оказываются чрезвычайно неясными и расплывчатыми. Они не сильно превосходят наборы графических примитивов. Каждый может понимать их, как ему угодно. Поскольку они бывают лишены всякой содержательности, большинство разработчиков относится к ним с полным презрением. И поскольку большинство разработчиков не считают своей обязанностью точно отображать создаваемые ими системы с помощью графики, мы заходим в тупик.

Наши рисунки часто оказываются бессодержательными, потому что каждый создает их по-своему. Разные люди используют свои символы для обозначения классов, объектов, подпрограмм, функций, баз данных – можете продолжить сами. Если нет стандартной системы обозначений, то каждый отстаивает свою; это выглядит, как если бы все говорили на одном языке, у которого множество диалектов, а наличие диалектов ведет к взаимному непониманию. Графическое представление в целом достаточно произвольно, поэтому не может быть «прав» кто-то один. Тем не менее люди защищают «свою» систему обозначений подчас с почти религиозным пылом. Разнообразие стилей задержало принятие графической нотации. Вот такая существовала проблема.

На самом деле под ней скрывалась другая, еще более глубокая. В отсутствие стандартных графических обозначений нельзя было наделить рисунки семантикой. Что я имею в виду под *семантикой*? В некотором смысле семантика и есть действительное содержание. Но сначала глубже разберемся с проблемой обозначений.

Как рассказывать об UML

Для того чтобы проверить, разобрались ли вы в некоторой системе понятий, попробуйте объяснить ее тому, кто не искушен в этой области. Для тех из нас, кто ежедневно занимается техникой, самый устрашающий вариант такой проверки – это необходимость изложить свое понимание обычным гражданам, т. е. неким разумным людям, которые связаны с техникой слабо или вообще никак. Сделать это очень трудно, потому что нельзя перейти на технический жаргон – стенографическую систему для высокоскоростного общения с коллегами, которая одновременно создает барьер между вами и непосвященными.

На практике я обнаружил, что разработчикам ПО бывает трудно объяснить нюансы своего ремесла другим профессионалам-инженерам. Во время поездки в Китай мне пришлось объяснять, что такое универсальный язык моделирования (Unified Modeling Language – UML) и каково его значение, техническим менеджерам, которые не были профессионалами в программировании. Я не ожидал, что такое препятствие возникнет, но когда я впервые произнес «UML», то увидел обращенные ко мне непонимающие взгляды. Я не мог двигаться дальше, не объяснив им, что такое UML. Но как это сделать?

Ниже следует 10-минутная презентация, которую я быстро сочинил, а впоследствии отшлифовал.

Что такое UML и в чем его значение?

Начнем с простого примера. В каком бы уголке света я ни написал на доске:

$$1 + 1 =$$

все поймут, что я хочу сказать. Как правило, кто-нибудь из присутствующих всегда выступит и скажет «2!». После этого я завершаю равенство:

$$1 + 1 = 2$$

и объясняю, что нас не только понимают всюду в мире, но и обычно подсказывают правильный ответ.

Это хороший пример *универсальной системы обозначений* – в данном случае числовой системы. Во всем мире люди пользуются ей для общения друг с другом. То, что напишет с ее помощью англичанин, поймет китаец, разговаривающий на мандаринском наречии.

На первый взгляд этот пример кажется тривиальным, но в действительности он демонстрирует поразительный факт: числа универсальны, а такие символы, как $+$ и $=$, всюду имеют одинаковый смысл.

Другая примечательная особенность этого примера состоит в том, что понять и оценить его может любой, кто окончил начальную школу. Его печальный недостаток заключается в том, что он выглядит более тривиально, чем оно есть на самом деле.

Второй, менее тривиальный пример

Мне пришлось согласиться, что этот первый пример, вероятно, излишне прост. Поэтому я начертил на доске треугольник (рис. 6.1).

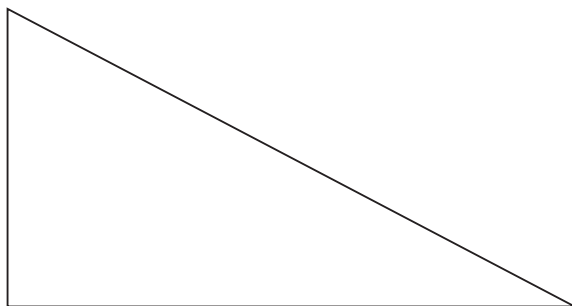


Рис. 6.1. Треугольник

Затем я отмечаю, что этот треугольник приобретает новое значение, если я дополню чертеж, как на рис. 6.2.

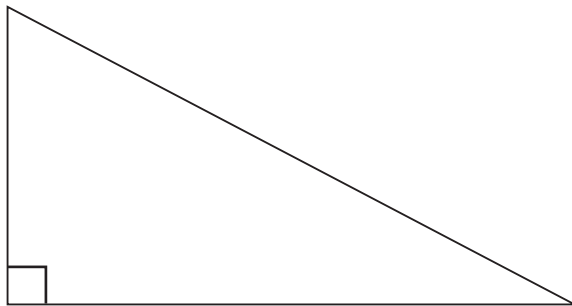


Рис. 6.2. Прямоугольный треугольник

Теперь этот треугольник вне всякого сомнения оказывается *прямоугольным*, потому что маленькая квадратная штукавина по общепринятому соглашению означает прямой угол. Кроме того, я теперь могу пометить стороны треугольника буквами А, В и С, как на рис. 6.3.

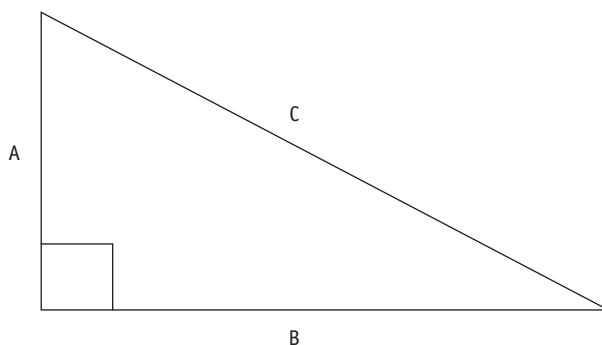


Рис. 6.3. Поименованный прямоугольный треугольник

И теперь я могу написать, что

$$A^2 + B^2 = C^2$$

Теперь у моего примера появилось несколько поучительных особенностей. Во-первых, это опять пример универсальной нотации. Прямые углы, прямоугольные треугольники и представляющие их символы всюду одинаковы, и в принципе с помощью таких схем древний египтянин мог бы обсуждать прямоугольные треугольники с современным перуанцем. Кроме того, начерченная схема прямоугольного треугольника определяет соотношение между его сторонами А, В и С. Теперь А, В и С не могут быть абсолютно произвольными величинами: если заданы любые две из них, то определена и третья. Эта схема включает в себе теорему Пифагора. Можно даже утверждать, что у этой диаграммы есть некая «семантика», что существует понятная связь между картинкой и значениями, выраженными буквами.

Действительно удивляет в этом примере то, что его может понять любой, кто окончил среднюю школу. Если кто-то ходил на уроки геометрии, то видел треугольники и прямоугольные треугольники, и если у него и осталось что-то в памяти от геометрии, так это старая добрая теорема Пифагора.

Итак, у меня теперь есть схема с семантикой, и я поднялся в уровне абстрагирования «ценой» перехода от математики начальной школы к мате-

матике старших классов. Кроме того, теперь мои слушатели явно заинтересовались, к чему же я клоню. Поэтому я цепляю для них на крючок вкусную наживку.

Третий пример

Приведенные выше примеры демонстрируют пользу универсальной системы обозначений. Проблема в том, что оба они взяты из мира математики, а математика, хотя имеет конкретные приложения, абстрактна по своей сути. А можно ли привести пример *не* из области математики?

Я нарисовал на доске рис. 6.4.

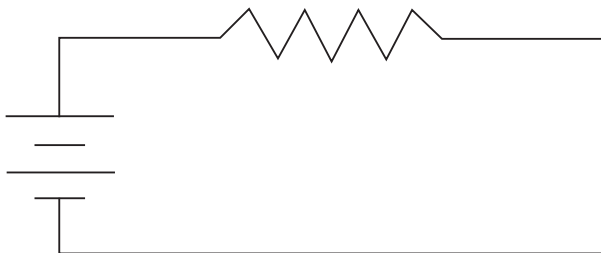


Рис. 6.4. Первая электрическая цепь

Поразительно в этом рисунке то, что, как только я изображаю его и сообщаю, что это простая цепь с батареей и резистором, головы поднимаются. Да, наверное, это простейшая электрическая цепь, которую можно изобразить, но тем не менее. Точно так же, как публика довольна собой, узнав по первым нотам Пятую симфонию Бетховена, она удовлетворена, поняв что-то техническое. Не давая им времени на размышления, я быстро ввожу дополнительно значки вольтметра и амперметра, как на рис. 6.5.

И смелыми завершающими штрихами я показываю, что если батарея дает напряжение 6 вольт, а сопротивление резистора составляет 6 ом, то в цепи течет ток в 1 ампер, как показано на рис. 6.6.¹

Что такое шестивольтовая батарейка, люди знают: ее можно купить в магазине. И большинство вспомнит, пусть смутно, что электрическое сопротивление измеряется в омах. Поэтому, когда вы наконец пишете на схеме «1 А», т. е. в цепи протекает ток в 1 ампер (я даже указываю направление то-

¹ Пуций эффект достигается использованием значка, а не слова «ом».

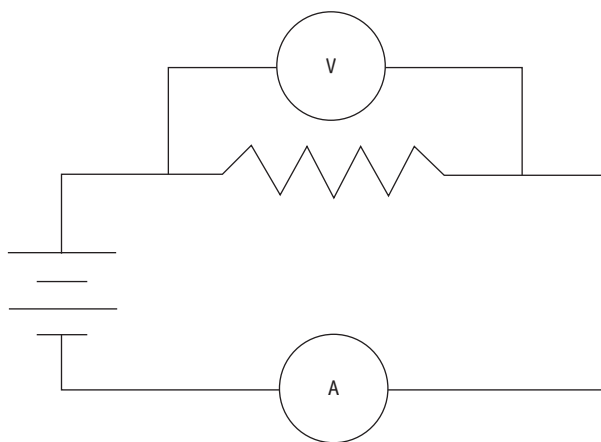


Рис. 6.5. Вторая электрическая цепь

ка!), все совершенно уверены, что знают, о чем вы говорите, даже если не помнят закон Ома.

Самое время отметить, что для студента из Швеции и радиолюбителя из Австралии эта схема имеет одинаковый смысл, хотя они говорят на разных языках. Стандартная международная система обозначений снова пришла на помощь. Но на этот раз схема не чисто математическая – у ее объектов есть реальное физическое воплощение. Кроме того, вступает в игру

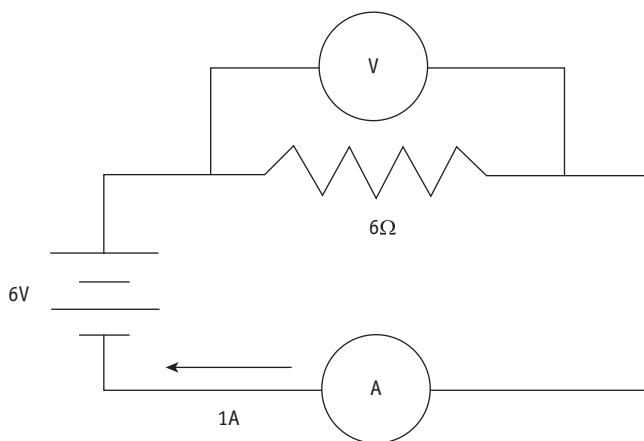


Рис. 6.6. Третья электрическая цепь

семантика: подразумевается не только закон Ома, но и направление тока, основанное на наших представлениях о положительном и отрицательном полюсах батареи, изображенных длинной и короткой горизонтальными линиями. Обычно я на некоторое время останавливаюсь, чтобы подчеркнуть обилие информации, передаваемой такой простой схемой, и трудности, с которыми столкнулась бы электротехника, не будь такой универсальной для всего мира системы обозначений.

Кстати, я уже переместил порог восприятия до уровня тех, кто прослушал годичный курс по основам физики.

А теперь вспомним о программах...

После этого наступает время подытожить и объяснить, какой прогресс достигается во всех областях благодаря стандартной системе обозначений, с помощью которой можно выражать понятия, и как схемы приобретают точность и смысл, когда к рисункам присоединяется семантика. Самые полезные из этих систем обозначений понятны в любой точке света.

Однако до 1996 г. не существовало стандартной системы обозначений для ПО. Пока UML не стал международным стандартом, два разработчика, *даже разговаривая на одном языке*, не имели возможности обсуждать свое программное обеспечение. Не было общепринятых соглашений относительно описаний ПО. Не удивительно, что это тормозило прогресс!

С приходом UML у разработчиков программ появился стандартный графический словарь для описания программ. Они могут чертить все более и более сложные схемы, представляющие их программы, так же, как электронщики могут рисовать все более и более сложные схемы, представляющие их электрические цепи. Допускаются вложенные схемы, поэтому можно отображать разные уровни абстракции.

Вклад Rational Software в этой области был огромен. Разработать UML и заставить крупнейшие мировые компании, такие как IBM, Microsoft, HP, Oracle и другие, согласиться с ним, было огромным шагом. Принятие UML в качестве стандарта международным органом – группой управления объектами (Object Management Group) – стало формальным процессом, необратимо изменившим положение. Строительство Вавилонской башни прекратилось, и все согласились с тем, как надо описывать программы.

Итак, значение UML установлено, и можно двинуться дальше.

Выходим на новый уровень абстракции

Конечно, сам по себе UML являет пример «технического жаргона». Пользуясь им, профессиональные разработчики ПО обсуждают между собой программы. По мере того как некая система обозначений становится глубже и полнее, она может стать утонченным способом выражения очень содержательных и сложных идей и конструкций, доступным только для посвященных. Тем не менее в начале эта (и любая другая) система обозначений на высшем своем уровне абстракции полезна для общения профессионалов с «населением». Это происходит благодаря тому, что с помощью основных элементов все еще можно передавать основные идеи. Действительно хорошая система обозначений допускает «вложенность» и множество уровней абстракции: высшие уровни способствуют общению людей, максимально далеких друг от друга в смысле подготовки и контекста, тогда как низшие уровни (технически самые детализированные) способствуют общению тех, чье понимание предметной области максимально близко, а именно технических специалистов.

Любопытно в нашем обзоре то, что для объяснения сути системы технических обозначений я обратился к аналогии. Мне удалось обойти западную «самоссылку», т. е. я объяснил UML, не описывая самого UML. Я описал жаргон, не обращаясь к жаргону. Сначала это покажется уверткой («послушайте, мне же не показали ни одной диаграммы UML!»), но на самом деле уметь описать предмет, не пользуясь им самим, необходимо.

В противном случае «население» окажется не в состоянии воспринять первую же диаграмму, которую вы начертите. Благодаря же такому вводному контексту, я считаю, первая UML-диаграмма будет воспринята гораздо лучше. Аудитория вернется к «1+1», теореме Пифагора и закону Ома и поймет, что то же самое вы делаете для программных конструкций.

Резюме

UML действительно стал *lingua franca* для описания ПО. Это хороший пример того, как вместо попытки сохранить закрытый стандарт можно отказаться от интеллектуальной собственности с целью дать толчок принятию отраслевого стандарта. В результате несколько компаний смогли заняться созданием инструментов для поддержки UML. Никому не пришлось создавать собственный стандарт, и конкуренция возникла на уровне инструментов, а не на уровне системы обозначений. Это в свою очередь

принесло ту пользу, что диаграммы, созданные разными инструментами, были понятны, поскольку в них была задействована одинаковая система обозначений и семантика. В большинстве сфер человеческой деятельности это принято называть прогрессом.

Все так. Но разработчикам все равно приходится писать код. Может быть, в меньшем объеме, если они благодаря помощи своих UML-диаграмм станут многократно использовать стандартные модули. Тем не менее код – это первооснова. От его написания никуда не денешься.

Так же, боюсь, как и от суеты, порождаемой успехами в создании языков программирования. Языки становятся все лучше и изощреннее, но они обречены на моральный износ. Разработчикам приходится изучать новый язык примерно раз в 10 лет, и это очень разрушительный процесс. Те, кому не удастся перескочить на очередное «великое изобретение», обречены на сопровождение старого кода, написанного на одном из прежних языков. Это вариант Чистилища, существующий специально для разработчиков программ.

Для нас, менеджеров, есть несколько иное препятствие. Дни, когда мы писали код, миновали, но развитие языков продолжается. Как же нам освоиться с новейшим языком? Это будет темой следующей главы.

ГЛАВА СЕДЬМАЯ

Написание кода



Рано или поздно все сводится к коду. Как бы ни хотелось избавиться от этого противного этапа, нельзя создать программное обеспечение, не написав код. Даже если вам кажется, что можно собрать приложение из готовых блоков, все равно придется написать код, который свяжет их все вместе и будет передавать результаты из одного модуля в другой.

Это и благо и несчастье одновременно. Благодаря хорошему коду наши системы становятся устойчивыми (robust) – в них не возникают отказы – и производительными (high-performance) – они быстро реагируют на наши требования. Благодаря хорошему коду мы выпускаем на рынок лучшие продукты, дающие нам преимущество в конкурентной борьбе. Откуда же берется тот ужасный код, который навязывают потребителям? Пусть тот, кто думает, что «код – он и в Африке код», сравнит материалы, написанные для «USA Today» и для «Economist». Те и другие написаны в общем-то на одном языке, но отличия в уровне, качестве и стиле очевидны. Сказать, что «английский – он и в Африке английский», значит не понимать существующих между ними различий.

Как еще говорят профессионалы, «дрянь можно написать на любом языке». Это ставит менеджеров программных разработок перед проблемой. В большинстве своем они сами когда-то писали код и, наверное, неплохой: часто благодаря этому они продвигаются по служебной лестнице. Но вполне вероятно, что они работали с другим языком программирования, а не с тем, который принят в их текущем проекте. В результате возникает то, что можно назвать *разобщиением первого уровня*: несмотря на то,

что менеджер разбирается в общих вопросах и задачах программирования, он может не вполне понимать, какие новые и интересные возможности открывает выбранный *модный* язык.

Но эта проблема несравнима по сложности с той, которая возникает у менеджера программного проекта в общении со *своим* менеджером, который наверняка не написал в своей жизни ни строчки кода. Объяснять ему, что проект затягивается из-за бага в коде – это все равно, что сказать, что его машину нужно отправить в ремонт, потому что в ней сломался кирпич. И даже хуже того, потому что автомеханик, наверно, скажет, сколько времени нужно, чтобы починить или заменить кирпич, а мы такого прогноза, увы, дать не можем. Когда в программе ломается кирпич, это может быть надолго.

Признаюсь, что у меня нет решения для второй проблемы. Я могу лишь попытаться изложить ситуацию с предельной ясностью. Зато по поводу решения первой проблемы – менеджера и «нового языка» – у меня есть некоторые рекомендации. Мое решение простое: менеджер программного проекта должен постараться познакомиться с новым языком на практике.

Как менеджеру изучить новый язык программирования

Раз в несколько лет на сцене появляется новый язык программирования, который обещает воплотить все наши робкие мечты и превзойти все языки, созданные до него. Разработчиков поопытнее шарахаются от таких новинок, как черт от ладана. Они слишком хорошо знают, что все языки более или менее похожи, и не обращают внимания на шум, поднимаемый вокруг каждого такого новшества.

С другой стороны, всегда есть надежда: вдруг на этот раз золота окажется больше, чем мусора. Вдруг кто-то изобрел более удачный цикл «do». Дело в том, что если мы неустанно ищем возможности усилить свои позиции в конкурентной борьбе, то нельзя просто отмахнуться от такого нового предложения. Приходится стиснуть зубы и снова бросаться на штурм.

Но как распознать действительную ценность нового языка, не тратя на его изучение слишком много нашего драгоценного времени? Чтение книг по новым языкам редко оказывается вдохновляющим занятием. Честно говоря, в большинстве своем они ужасны. Иногда обнаруживаются жемчужины, которые не только порождают новое поколение программистов,

стов, но и выдерживают проверку временем; например, *K&R*¹ стала классическим руководством по программированию на С с тех пор, как этот язык приобрел популярность. Книга служит для первого знакомства с С и содержит все, что требуется от справочного руководства, в виде краткого, доступного для понимания и недорогого (когда-то) издания в бумажной обложке. (Сегодня эта тоненькая книжечка стоит \$40.) Но, как я сказал, такого рода краткие, хорошо организованные и высокоинформативные книги являются редкостью.

Поэтому, посмотрев одну-другую книгу по новому языку из тех, что появились в продаже, большинство профессионалов начинает изучать новый язык с того, что пишет свою первую программу. Это более эффективный метод изучения, чем просто чтение надуманных и искусственных примеров, предлагаемых большинством книг для знакомства с языком. Чтобы освоить что-то, обычно очень полезно попытаться решить задачу.

Задача, более точная формулировка

Если вы решили написать программу, то нужно выполнить калибровку. С момента выхода *K&R* стало почти стандартом, что первая программа на новом языке всего лишь печатает или выводит на экран сообщение «Hello, world». Конечно, вы узнаете, как вывести строку и воспользоваться компилятором, компоновщиком и загрузчиком, но не более того. То, что вы получите в результате, нельзя даже рассматривать как первый проект.

Я считаю, что первая программа не должна быть такой тривиальной. Нет риска – нет и выигрыша. С другой стороны, нежелательно затевать большой проект, настолько сложный, чтобы для его реализации потребовалось изучить идиоматику нового языка, но не пришлось осваивать новую предметную область. Это значит, что нужна некая *стандартная задача*, так чтобы при каждой реализации ее решения средствами нового языка программирования, который требуется оценить, происходила ваша *калибровка*. Иными словами, нужно обойтись без создания новой науки и разработки новых алгоритмов. Теоретически это упражнение должно становиться более легким для каждого нового языка, поскольку

¹ Керниган и Ритчи (Kernighan, Brian W. and Dennis M. Ritchie), *The C Programming Language*, 2nd edition (Upper Saddle River, New Jersey: Prentice Hall, 1988); Брайан Керниган и Деннис Ритчи «Язык программирования Си». – Пер. с англ. – СПб.: Невский Диалект 2000.

ку проблематика оказывается для вас знакомой и можно больше времени уделить оценке того, насколько хорошо решение формулируется на новом языке. Если для решения стандартной задачи средствами нового языка требуется вчетверо больше времени, чем обычно, то могут возникнуть вопросы относительно характеристик нового языка и/или динамики его изучения.¹

Что должно быть в стандартной задаче?

Хорошо, что вы задали этот вопрос. Вот ряд вещей, которые я хотел бы исследовать во всяком новом языке:

- **Как вывести строку.** Это полезно уметь, например, чтобы показать пользователю приглашение для ввода данных. Как указывалось в предыдущем разделе, это как раз и демонстрирует программа **Hello, world**.
- **Как получить данные, введенные пользователем.** Можно начать с простых строк и двигаться к форматированным числам. Простое чтение символьных строк дает массу возможностей, поэтому такая задача может служить хорошей отправной точкой.
- **Простые алгоритмы.** Нужно поработать с какими-то данными. Не требуется ничего особенного, просто какие-нибудь присваивания, арифметические операции и т. д. При этом выяснится, потребуются ли математические библиотеки и т. п. Не обязательно проверять, выдержит ли оснастка вашего судна шторм, но спустить его на воду придется.
- **Как осуществляется постоянное хранение данных.** Это уже большой шаг вперед, потому что надо записать результат и сохранить его так, чтобы он не пропал после завершения программы. В идеале пробная программа должна и записать данные в постоянное хранилище, и считать их оттуда. Вообще говоря, это демонстрирует существующий в языке интерфейс к некой файловой системе. Достаточно ограничиться простым текстовым файлом и не усложнять задачу.
- **Как реализовать «стандартную» структуру данных типа связанного списка.** Необходимость в них регулярно возникает при реше-

¹ Конечно, может быть, что со временем мы утрачиваем свое мастерство. Но такую возможность согласится допустить только настоящие пессимисты.

нии технических задач программирования, поэтому полезно иметь такую структуру в учебной задаче, чтобы узнать, как она реализуется в новом языке. Этот пункт просто развивает «простые алгоритмы», о которых сказано выше.

- **Как обрабатывать ошибки.** Что делать, если данные, введенные пользователем, или в файле не те, на которые вы рассчитывали – искажены или просто дурацкие (или вообще отсутствуют)?
- **Как оценить возможности абстрагирования и инкапсуляции.** Насколько просто или сложно реагировать на изменение требований или постановку задачи?

Я хочу сделать важное *предупреждение*. Здесь исследуются возможности языка для «программирования на низком уровне». Несмотря на их важность, при этом не тестируются возможности языка для программирования «высокого уровня», например, возможность иметь открытые или закрытые интерфейсы, взаимодействие с другими программными инфраструктурами и, разумеется, графика. В частности, мощь последних пополнений нашего арсенала языков в значительной мере определяется богатством библиотек классов и т.д. Их можно оценить только за счет ощутимых дополнительных трат времени. Однако исследовать их будет проще, если вы разберетесь с базовыми методами программирования.

Игра в животных

Вот программа, которую я переписываю в качестве своей стандартной задачи с 1960-х гг.¹ Она называется «The Animal Game» (игра в животных).

Программа представляет собой интерактивный диалог между пользователем и программой. Пользователю предлагается «задумать животное». Затем программа начинает задавать вопросы: «Это животное – гончая собака?».² Если пользователь задумал гончую, то отвечает «да»; тогда программа хвалит себя за проницательность, благодарит пользователя за игру и заканчивает работу.

¹ Должен признаться, что горжусь тем, что выпускаю программы профессионального качества уже пять десятков лет, начиная с шестидесятых годов. В прежние времена я много писал сам, но потом мои коллеги очень старались, чтобы у меня не было возможности писать код. В течение этого промежутка времени мне пришлось работать с многими языками.

² Дань уважения Snoopy и его создателю Чарльзу Шульцу (Charles Shulz).

Если же пользователь задумал другое животное, он отвечает «нет». Удрученная программа сообщает: «Увы, я не угадала ваше животное. Назовите мне его и введите вопрос, на который нужно ответить „да“ для вашего животного и „нет“ для гончей».

Например, если человек задумал форель, он должен ввести «форель», а затем вопрос «это рыба?». Ответом будет «да» для форели и «нет» для гончей.

Побив программу и введя свое животное и свой вопрос, пользователь читает благодарность, а программа снова завершается.

Однако когда пользователь начинает игру в следующий раз, происходит нечто иное. После предложения «Задумайте животное» первым задаваемым вопросом становится «Это рыба?». Если ответ «да», программа спрашивает «Это форель?». Но если на вопрос «Это рыба?» ответом будет «нет», программа спрашивает: «Это гончая?». Если пользователь задумал форель или гончую, программа угадывает правильно и выигрывает. Если же пользователь задумал другое животное, программа признает поражение и просит ввести новое животное и вопрос, по которому его можно отличить от гончей или форели.

Таким образом, сначала программа совсем не блещет «сообразительностью». Но становится «умнее» в процессе игры, запоминая новых животных и новые вопросы. Она не всегда угадает ваше животное самым коротким способом, но через какое-то время она сможет изображать интеллект и «угадывать» ваше животное почти всегда. Это происходит потому, что с ростом ее базы данных она сможет «выслеживать ваше животное» все более и более уверенно.¹

Удовлетворяет ли игра в животных критериям?

Вполне. Напомню их:

• **Вывод строк:** программа должна давать пользователю указания и реагировать на ответы, которые он дает на ваши вопросы.

¹ Здесь есть одно слабое место. Пользователь может ввести плохой вопрос или другим способом устроить путаницу, например поменяв местами «да» и «нет». Не смейтесь, я с этим сталкивался. Однажды игрок в середине игры забыл, какое животное он задумал. Когда это происходит, база данных калечится. В этой игре я чаще добивался успеха с третьеклассниками, чем со взрослыми. Очевидно, простота игры идеальна для восьмилетних детей и обескураживает их родителей.

- **Ввод:** программа должна принимать от пользователя строки и что-то с ними делать. Диалоговый характер задачи требует некоторого (не слишком сложного) анализа введенных данных.
- **Простой алгоритм:** в зависимости от ответа программа выбирает разные вопросы. Таким образом, в зависимости от ответа «да» или «нет» выполняется обход структуры данных.
- **Взаимодействие с постоянным хранилищем:** программа должна где-то хранить текущую базу данных с животными и вопросами и считывать ее при запуске. Эти данные исчерпывают набор вопросов. Если программа не угадает животное, то должна будет обновить файл, записав в него новое животное и новый вопрос. Затем их надо сохранить для следующего сеанса работы.
- **Прототип структуры данных:** итак, чтобы получить результат, программе придется просмотреть связный список какого-то вида. Когда пользователь добавит новое животное и новый вопрос, программа должна будет добавить эти ссылки и обновить некоторые прежние, чтобы они указывали на новую информацию.
- **Обработка ошибок:** программа должна справиться с ситуацией, когда пользователь вводит пустые данные вместо требуемых реальных значений или файл данных оказывается поврежденным. Здесь большой простор для действий в зависимости от того, насколько дружелюбной пользователю вы хотите сделать программу. Например, как быть, если пользователь внезапно решит прекратить работу?
- **Абстракция и инкапсуляция:** задачу можно легко обобщить, сделав ее игрой в угадывание овощей, минералов или известных личностей. Эти варианты должны зависеть только от загрузки разных файлов данных: язык должен позволять делать все с помощью одной и той же программы.

Заметьте, что я все очень упростил. Например, нельзя разрешать играть в эту игру одновременно нескольким людям, иначе задача сильно усложняется. Но даже для последовательной работы нескольких пользователей это неплохая задача программирования.

Языки, прошедшие тест

С течением времени я реализовал «игру в животных» на следующих языках:¹

FORTTRAN
BASIC
APL
Pascal
FORTH²
C
Ada
C++

Следовало бы еще на Java, но...³

Обычно мне требуется несколько часов, чтобы вспомнить, как организовать связный список, и выдать что-то готовое. Для того чтобы другие смогли играть в эту игру без моего надзора (это значит, что надо организовать какую-то обработку ошибок), мне обычно требуется несколько дней программирования. Возможно, это получалось бы у меня быстрее, если бы я мог найти код, оставшийся от предыдущих опытов, но это мне никогда не удастся. Данное упражнение повторяется с периодичностью четыре-пять лет, которых достаточно, чтобы потерять предыдущий пример.

Вы не знаете, зачем нужен старый код? Разве нет поблизости проектной спецификации, документа на псевдокоде или диаграммы UML, с помощью которых можно написать реализацию, не заглядывая в прежний код?

¹ Этот перечень раскрывает мои программистские корни: я пришел из научной среды, а не из деловой. Хотя однажды мне и пришлось руководить группой CO-BOI-программистов, но я был тогда так занят, что изучение их языка оказалось слишком низко в списке приоритетных задач. И если быть абсолютно честным, в то время я все еще страдал «языковым снобизмом».

² FORTH составляет исключение из того правила, что если название языка состоит только из прописных букв, то это акроним. Чарльз Мур, изобретатель FORTH, хотел сделать его языком четвертого поколения, но компьютер, на котором он работал, допускал только пятисимвольные идентификаторы и заглавные буквы, — по крайней мере, так гласят предания. Отдельные буквы FORTH ничего не обозначают.

³ Я действительно становлюсь для этого слишком стар. Чтобы соблюсти полную справедливость, я поступлю так же и в отношении C#.

Ответ состоит из двух частей: да, я рисую диаграммы и пишу псевдокод, но мне приходится каждый раз делать это заново, потому что их я тоже не могу найти. Во-вторых, полезно посмотреть, как ты делал что-то на старом языке, когда нужно сделать то же самое на новом. Я бы предпочел иметь возможность найти старые диаграммы с псевдокодом *и* реализацию на старом языке. Все это вместе дало бы мне хороший начальный толчок. Если бы только я тщательнее вел учет. Ладно. Конечно, это вопиющее нарушение правил софтверной компании, но, может быть, не стоит преувеличивать размер этого греха, поскольку в сущности это лишь маленький личный программный проект?

Самые необычайные из моих реализаций были сделаны на APL и FORTH. Те, кто работает с этими языками, могут догадаться о причинах, а объяснять остальным бесполезно. В списке подозреваемых отсутствуют LISP, Smalltalk, PL/I и COBOL. Ничто не мешает вам попробовать в них свои силы. Мне этого сделать не пришлось.

Кстати, выполняя это упражнение, вы знакомитесь также с особенностями среды разработки и доступными инструментами. Я, например, выяснил, что первые отладчики C++ были не блестящего качества; напротив, практичность Rational Environment была очевидна, и моя реализация на Ada была осуществлена в рекордно короткое время. Занимаясь этой задачей на FORTH, я научился перезагружать свою машину при каждой ошибке этапа исполнения.

Это ваша игра

Лучший способ освоиться с новым языком программирования состоит в том, чтобы запрограммировать стандартную задачу, которая вам уже знакома. Это позволит вам изучить язык и узнать, «как это сделать» для известного набора практических вопросов. Моя стандартная задача служила мне долгие годы, и я считаю, что она заставляет решать минимальный, но полезный набор базовых проблем.

Резюме

Ограниченность данного подхода в том, что он сосредоточен на проблемах низкого уровня программирования. Каждый новый язык обладает большей выразительной силой, и часто эта сила не проявляется в маленьких примерах. Например, не видно, каким образом данный при-

мер способен помочь понять наследование в C++, хотя я уверен, что заинтересованный читатель найдет способ.

Мне неоднократно приходилось замечать, что большинство разработчиков работает с неким подмножеством новейшего языка, с помощью которого они и решают свои задачи. Лишь немногие эффективно применяют «мощные функции», перевозносимые в книгах и звонкой рекламе, произрастающих на новом языке. Меня также беспокоит, что эти языки настолько усложнены, что когда искушенные пользователи применяют самые передовые возможности языка, то код становится непонятен программисту средней квалификации. Это особенно опасно, если учесть, что сопровождением кода в большинстве случаев занимается программист, менее компетентный, чем автор первоначального кода. Это не критика современных языков, а рекомендация менеджерам, которых интересует, чтобы выпускаемый ими код можно было сопровождать в течение длительного времени.

Далее мы переходим к главе 8, последней в этом разделе, посвященном отличиям, присущим производству ПО. Речь пойдет о малоизвестном искусстве, называемом «как выставить продукт за дверь». Это то, за что нам платят деньги, и мне горько признаться, что часто мы не слишком хорошо с этим справляемся.

ГЛАВА ВОСЬМАЯ

За дверь его!

Иногда я говорил, что могу создать совершенный продукт. Если только не ставить мне условий о его поставке.

Как только вы потребуете, чтобы продукт был отгружен к определенной дате, я могу гарантировать, что продукт совершенным не будет. Он обязательно кому-нибудь чем-нибудь не понравится. У него не будет каких-то функциональных возможностей, в его работе будут незначительные, но докучливые неполадки, его документация будет неполной, и, разумеется, его пользовательский интерфейс будет местами шероховат. Если бы только у нас было больше времени...

Несовершенство не является уникальной особенностью только программных продуктов. Любой выпущенный в свет продукт всегда представляет собой компромисс – между тем, что мы хотели бы сделать в идеале, и тем, что мы вынуждены выпустить, чтобы начать получать доход. И, хотите – верьте, хотите – нет, иногда отгруженный продукт и в самом деле оказывается *достаточно хорошим, хотя и представляет собой компромисс*. Для этого он должен удовлетворять одному условию: максимально удовлетворять запросы большинства пользователей.

Рассмотрим, например, *выпуск новой версии* программного продукта – версии, которая предоставляет некоторые новые возможности и в которой устранено множество раздражающих дефектов. Можно работать над этой версией бесконечно. Чем дольше, тем больше удастся добавить новых возможностей и исправить недостатков. Но можно взглянуть на это иначе:

чем дольше вы станете задерживать отгрузку новой версии, тем дольше ваши клиенты будут вынуждены уживаться с проблемами той версии, которой сейчас пользуются. Таким образом, вопрос ставится так: что лучше – избавить пользователей от 50 дефектов сегодня или от 55 дефектов, но через две недели? Если тысячи пользователей ежедневно страдают от дефекта №29 в вашем списке, то, я думаю, есть достаточно оснований утверждать, что отгружать продукт *нужно было еще вчера*.

Как только вы начинаете понимать, что отгрузка продукта – это не просто часть вашей работы, а весьма критичный момент в проекте – «пересечение финишной ленты», как сказал бы Роберт Бонд,¹ – вы должны подумать обо всем том, что требуется для превращения скопища работающих битов в красиво упакованный продукт, который можно выставить на полку магазина, или в набор файлов, которые можно выложить на сервер. Надо подумать о тестировании, средствах инсталляции, документации, организации поддержки пользователей и многом-многом другом. Это чрезвычайно болезненный процесс, который нужно проделать сотни раз, чтобы привыкнуть. Это как смерть от тысячи мелких порезов. По крайней мере до тех пор, пока вы не пройдете через это первую сотню раз или что-то около того. Отгрузка продукта – это одно из тех упражнений, которые требуют методичности и настойчивости в доведении всего до конца – вплоть до самой последней мелочи.

В этой главе я вовсе не собираюсь до смерти замучить вас банальностями. Я собираюсь сфокусироваться на одной небольшой части проблемы: как «закруглиться» с разработкой программного продукта, чтобы в конце концов суметь его отгрузить? Что должно измениться при «заходе на посадку»? Мой ответ таков: если вы заблаговременно сделали все правильно, то никакого изменения практически не заметите; если же вы старались не думать о предстоящей отгрузке, то в конце вам придется пережить суровый и разрушительный катаклизм, а ваша способность выдать продукт на гора подвергнется великой опасности.

¹ Роберт Бонд (Robert Bond) на протяжении многих лет возглавлял отдел продаж и маркетинга в Rational Software. Для меня он также был очень хорошим наставником.

Если Вы его постройте, они придут¹

Мир программных продуктов неоднократно был свидетелем успехов и неудач, определявшихся законами свободного рынка. Но список провалов будет неполным, если мы не включим в него те проекты, чьи детища так никогда и не увидели дневного света – проекты, над которыми велась работа, но которые так и не завершились отгрузкой продукта. Как ни очевидна эта истина, но вы не можете достигнуть успеха, не пройдя через процесс отгрузки.

Невозможно отгрузить то, чего не существует, поэтому так важно собрать продукт из множества его частей. При этом процесс сборки повторяется многократно. Вы будете собирать продукт *снова и снова* – до тех пор, пока один из ваших *кандидатов на выпуск* не пройдет контроль качества, и вы не примете решение о его поставке.

Пора взглянуть в лицо тем проблемам, которые возникают при организации периодической сборки продукта.

Вначале была песочница

Продукты порождаются проектами, а проекты имеют тенденцию начинаться в хаосе. В организациях, где налажены четко определенные процессы, каждый разработчик трудится над своей частью продукта в относительно изолированной обстановке, которую иногда называют *песочницей*. При этом создается какой-нибудь механизм, зачастую по типу *ad hoc* для сборки всех частей, выходящих из песочниц, так чтобы каждая группа разработчиков могла тестировать результаты своего труда в контексте всего продукта. Системы управления конфигурацией позволяют организовать работу так, чтобы разработчики не наступали друг другу на ноги и могли работать автономно, в то же время обеспечивая контекст для свободной интеграции.

Такой нестрогий подход работает вполне удовлетворительно на самом раннем этапе проекта, когда все быстро меняется, когда архитектура еще

¹ На самом деле фраза в фильме «Field of Dreams» («Поле грез») звучала так: «Если вы его постройте, он придет». При этом персонаж, произносящий эту фразу, под словом «он» подразумевает то ли Босого Джо Джексона, то ли отца главного героя. Эту репризу искажали столько раз, что теперь все говорят именно «они». И правда, в конце фильма на построенное бейсбольное поле действительно пришли «они».

четко не определена и интерфейсы еще не закреплены. Однако даже скромные по своему размаху проекты довольно быстро перерастают такую инфраструктуру. И тогда происходит одно из двух: либо организация начинает заниматься *сборкой* серьезно и систематически, либо она этого не делает. Те, кому удастся наладить «сердцебиение» проекта – регулярный и надежный цикл сборки, – увеличивают свои шансы на успех. Те же, кто не удосуживаются установить такой ритм, вскоре замечают, что энтропия начинает одерживать верх и с течением времени сборка продукта становится все более и более трудной.¹

Во многих организациях крайне недооценивают усилия, необходимые для учреждения добротного процесса сборки. Поэтому на более поздних этапах проекта его участники зачастую сталкиваются с «новой» проблемой: им приходится бороться не только с многочисленными дефектами, незавершенными компонентами и тому подобными осложнениями, но еще и с тем, что до сих пор принималось как должное – со сборкой своего продукта воедино. Западня для неосторожных. Чтобы не угодить в нее, необходимо лучше понимать, в чем состоит процесс сборки продукта.

Почему же сборка продукта так трудна?

Прежде всего надо сказать, что в продукте, который вы собираетесь отгрузить, частей больше, чем в тех прототипах, которые вы мастерили для внутреннего потребления. В качестве классического примера можно взять справочную систему. Разработчики и тестеры заглядывают в нее редко, поскольку и так знают продукт достаточно хорошо для выполнения своей работы. Поэтому прежде чем продуктом попробуют воспользоваться «посторонние», необходимо приложить определенные усилия, чтобы гарантировать его полную работоспособность. Затем надо подготовить инструкции по установке продукта на различные системы, а также прочие материалы и принадлежности, без которых вы сами прекрасно можете обойтись. Поэтому первая существенная проблема, с которой вы сталки-

¹ *Энтропия* – это тенденция всех систем переходить из упорядоченного состояния в беспорядочное при отсутствии внешних воздействий. Это фундаментальный закон физики. Можно также сказать, что для любой цивилизации любые попытки совершать прогресс представляют собой попытки бросить вызов энтропии. Можно сказать то же самое и иначе: превращение хаоса в порядок требует работы, и как только такая работа прекращается, система будет самопроизвольно двигаться в направлении беспорядка. Я еще вернусь к этой теме в главе 18.

ваетесь при организации процесса сборки – это так называемая упаковка (packaging). Для поставки продукта требуется больше составляющих, чем для работы с ним внутри вашей организации. Кроме того, надо не забыть задокументировать множество мелких деталей, которые были всегда известны членам команды или воспринимались как нечто само собой разумеющееся.¹ Работу по подготовке продукта для «посторонних» иногда называют «шлифовкой». Если вы сразу же не устраните «заусенцы» (по крайней мере, самые «острые» из них), ваши первые пользователи могут о них сильно «порезаться».

Допустим, однако, что это всего лишь проблема из области логистики и при надлежащем планировании она не сорвет ваши графики. В известном смысле можно считать ее «неприятной необходимостью»: если вы будете ее игнорировать, она вас «ужалит»; если же вы будете знать об этой проблеме и готовиться к ней заранее, то решить ее будет не так уж трудно. Предупрежден – значит, вооружен: относитесь к упаковке, как к чисто технической проблеме, и все у вас будет в порядке.

На самом деле на пути к успешному процессу сборки стоят три гораздо более фундаментальные проблемы. Их следует различать, но при этом они тесно взаимосвязаны и должны быть решены для достижения успеха.

Препятствие первое: организационная политика

Многие менеджеры разработки ПО упускают из вида тот факт, что управление процессом сборки является прежде всего политической проблемой. Выражаясь простым языком, тот, кто контролирует процесс сборки, обладает огромным влиянием. Как-никак цикл сборки определяет ритм разработки и тестирования в целом, и его можно уподобить сборочному конвейеру на фабрике. Тот, кто определяет режим работы конвейера и в частности его скорость, в значительной степени определяет выпускную производительность фабрики. Все, кто стоит у конвейера, прекрасно осознают свою подчиненность его ритму. Замедлить этот ритм или, боже

¹ Одно из традиционных названий документации такого рода – «*примечания к выпуску*». В примечания к выпуску обычно включают описания ограничений выпускаемой версии продукта, известных дефектов и т.д. Иными словами, делается попытка описать состояние продукта на момент отгрузки. Лучше рассказать клиенту обо всех тех важных деталях, которые вам известны, чем устраивать ему ненужные сюрпризы. Иногда примечания к выпуску помещают в файл с названием *readme*.

упаси, остановить конвейер совсем – огромный грех.¹ В индустрии разработки ПО эквивалентом остановки конвейера является внесение в код таких изменений, которые приводят к *сбою сборки*.

Теперь обратим внимание на то, что в процессе сборки принимают участие все, но контролируют его только некоторые. По своей природе сборка – это недемократическое мероприятие, и для того чтобы она работала, требуется определенный иерархический организационный аппарат. И хотя все с этим более или менее согласны, есть одна загвоздка: надо определить, на кого будут возложены ответственность и полномочия по обеспечению функционирования сборки – ведь с момента своего назначения эта группа лиц обретет немалое влияние и реальную власть.

Поскольку люди не склонны расставаться с властью и влиянием, процесс сборки превращается в политическую игру. Это порождает бессчетные дискуссии о том, кто и какие будет иметь права и обязанности в интересах процесса сборки. Во время таких дискуссий обнажаются все негативные политические тенденции в организации.

Ревнителю чистоты воскликнут, что политические устремления достойны только осуждения и им надо всячески препятствовать: работа, мол, достаточно трудна с технической точки зрения и без «загрязнения» ее политикой. Однако в большинстве организаций *желание устранить* политику не обязательно приводит к ее *реальному устранению*. Политика – это неотъемлемая часть окружающего нас реального мира, и с ней приходится иметь дело.² Вы должны пройти через этот этап, каким бы неприятным он ни казался. Иначе вы не справитесь с последующими двумя препятствиями.

Вот несколько более конкретных советов:

¹ Конечно же, существуют уважительные причины остановки конвейера, и работник конвейерной линии иногда является именно тем человеком, кто должен нажать красную кнопку. Тем не менее такое действие, как остановка конвейера по ошибке, безусловно, является недопустимым.

² С моей точки зрения, политика политике рознь. С одной стороны, есть «хорошая политика» – нечто сродни понятию «честной борьбы» – и здоровый политический процесс может и должен помогать принимать правильные решения. С другой стороны, существует «плохая политика», в результате которой цели и действия организации могут подчиниться карьерным интересам отдельных личностей, и политику такого сорта необходимо затаптывать на корню. И конечно же, есть еще проблема «серой зоны» – зоны между «хорошим» и «плохим». Я коснусь этой темы более подробно в главе 13 «Политика».

- Попробуйте убедить группу, что кто-нибудь должен быть назначен главным, поскольку свободная конфедерация обречена на провал.
- Попробуйте найти разумный компромисс между автономией отдельных участников и полномочиями централизованной власти.
- Обязательно добейтесь того, чтобы руководство понимало важность данной проблемы и выделило для сборки лучших людей.
- Далее в этой главе мы будем говорить о *царе сборки*. Добейтесь того, чтобы эту роль исполнял человек технически компетентный, решительный, справедливый и уважаемый. Введите его в эту должность как можно раньше и поручите ему штурвал на политическом мелководье.
- Заручитесь поддержкой руководства в деле подавления «плохой политики» в случае, если таковая высунет свою безобразную морду.

Препятствие второе: процесс

Продравшись сквозь политические джунгли вокруг учреждения группы, контролирующей сборку, все участники должны теперь договориться о процессе, который будет осуществляться. Подобно тому, как форма обычно следует за содержанием, процесс очень часто отражает те политические компромиссы, которые были заключены на данный момент. Политика и процесс довольно интенсивно взаимодействуют друг с другом. Зачастую сложности с процессом проявляются рано – уже на первом этапе, поскольку он используется как суррогат теми, кто не хочет открыто признать существование нерешенных политических разногласий. В некоторых организациях можно наблюдать переплетение политики и процесса в один гигантский клубок, компрометирующий репутацию «процесса». Нельзя использовать «процесс» для решения сугубо политических проблем – точно так же, как невозможно «решение» технических проблем посредством политических компромиссов.

Основной конфликт возникает между сторонниками строгого, жесткого процесса (иногда характеризуемого как «*множество правил и никакой милости*»)¹ и теми, кто предпочитает относительно свободные порядки.

¹ По-видимому, мы обязаны этой характеристикой Джеймсу Арчеру (James E. Archer). Джеймс – один из самых эффективных и успешных менеджеров разработки ПО, которых я знаю. Он стоял у истоков и был крестным отцом целого семейства программных продуктов, разрабатывавшихся в Rational Software, – сред программирования. Мы с ним провели множество интересных дискуссий о «правильном дозировании» процесса.

Обычно лучший первый шаг в таком положении состоит в том, чтобы заставить всех признать тот факт, что единственного и правильного ответа на данный вопрос не существует. Каждая организация оригинальна и своеобразна, и поэтому ваш процесс должен быть настроен на особенности именно вашей организации.¹

Это не означает, что надо изобретать процесс заново. Я неспроста употребил слово «настроен» в предыдущем абзаце: я твердо убежден, что лучший способ решения данной проблемы – это отталкиваться от известного базового процесса, работоспособность которого уже была продемонстрирована в прошлом. Например, разработанный в Rational Software процесс «унифицированного управления изменениями» (Unified Change Management, или UCM) имеет богатую историю успешного применения на практике. Мы знаем, что он годится для широкого спектра программных продуктов, предметных областей и организаций. Зачем начинать все сначала? Уверены ли вы, что сможете сделать лучше?

Существует несколько типичных ловушек, в которые вам надо постараться не попасть при обсуждении процесса. Первый печально известный капкан такого рода – это так называемые *религиозные войны*. Во всякой организации есть гурь, которые свято верят, что они (и только они) знают магическую формулу «правильного» процесса. И почти наверняка в этой же организации найдутся несколько членов оппозиции, так же свято убежденных в своей правоте.² Вне зависимости от того, кто оказывается прав, эти крестовые походы абсолютно непродуктивны и зачастую вращаются вокруг смутных и малозначительных деталей. Подготовленный менеджер должен как можно раньше выявлять религиозных фанатиков и подавлять их. Иногда не остается ничего иного, как потребовать, чтобы они замолчали. Всегда помните, что процесс не является самоцелью; настоящая

¹ В этом месте некоторые люди возражают, что сначала надо попытаться выработать «правильный» процесс, а уже затем подстраивать под него организацию. Что ж, это достойная похвалы цель, и *теоретически* такой подход правилен. Однако я редко бывал свидетелем успеха этого подхода на практике. Недопустимо позволять консервативной организации отвергать разумный процесс, но с другой стороны, реализация процесса, слишком «передового» для данной организации – дело весьма затруднительное.

² Чтобы проиллюстрировать, как далеко могут зайти религиозные войны, достаточно вспомнить случаи, когда противоборствующие стороны называли друг друга «нацисты от процесса» и «анархисты». Когда в дело идут подобные ярлыки, очень трудно вести конструктивные дискуссии.

цель – это выпуск продукта, а процесс – всего лишь одно из средств достижения этой цели!¹

Другая известная ловушка – это вера в то, что процесс освобождает от необходимости мыслить и принимать решения. На любое правило, даже самое железное, обязательно найдется исключение. Какую бы процедуру вы ни учредили, вам придется внимательно наблюдать за происходящим и при необходимости корректировать курс. Как я уже отмечал, вы будете подлаживать и подстраивать ваш процесс на ходу по мере выяснения, что в нем пригодно для вашей ситуации, а что нет.

Наконец, не затягивайте установление процесса. Лучшее – враг хорошего.² Процедуру надо разрабатывать итеративно – точно так же, как программное обеспечение. Приступайте к итерации номер 1 как можно скорее. Учитесь. Изменяйте. Улучшайте. Повторяйте, пока не закончите.

Препятствие третье: инструментарий

Первое и второе препятствия – политика и процесс – очень тесно связаны друг с другом. То же самое можно сказать о взаимоотношениях второго и третьего. Третье препятствие – это, конечно же, тот самый инструментарий, который вы будете применять для реализации процесса. Казалось бы, не надо упоминать, что начинать с выбора инструментария все равно, что ставить все с ног на голову, но, к моему великому удивлению, именно так и поступают во многих организациях. Когда процесс определяется инструментами, может получиться забавный результат, если окажется, что он идет вразрез с политической философией организации.

Очевидно, что вам необходимы именно такие инструменты, которые помогут проводить в жизнь выбранный процесс. Если процесс позволяет

¹ Аналогичным анархистам было бы трудно продемонстрировать, что они могут поставить продукт, не соблюдая никакого процесса. Как это часто бывает в подобных спорах, крайние позиции оказываются уязвимыми.

² Впервые я услышал это высказывание от Михаила Драбкина из Риги (Латвия) и решил, что оно представляет собой русскую народную поговорку. Возможно, он цитировал (неточно?) советского адмирала Сергея Горшкова. Доктор Стефен Фрэнклин (Stephen Franklin) из Калифорнийского Университета города Ирвайн (UC Irvine) отмечает, что похожие высказывания приписывают одновременно Клаузевицу и Вольтеру. При этом высказывание Клаузевица более многословно, а Вольтер, возможно, «позаимствовал» свой вариант из итальянской пословицы. Происхождение подобных лаконичных выражений порой очень трудно установить, но это никак не умаляет ни истины, ни мудрости, в них заключенных.

делать ошибки, вы будете время от времени «откатывать» изменения. Поддерживает ли ваш инструмент такие «откаты»? Если разработчики территориально удалены друг от друга, будут ли они регулярно сдавать свои фрагменты измененного кода в общее централизованное хранилище для поддержки базовой версии продукта? Если да, то лучше, чтобы ваш инструмент был приспособлен к такой модели. Намерены ли вы каждую ночь собирать продукт с начала и до конца? Если да, то я надеюсь, что производительность вашего инструмента позволит это делать. Хотите ли вы автоматически производить регрессивное тестирование при каждой сборке продукта? И в этом случае поддержка со стороны вашего инструментария необычайно важна.

Даже те организации, которые хорошо справляются с первыми двумя препятствиями, иногда спотыкаются на третьем. И в этом не всегда виноваты инструменты. Возвращаясь к аналогии с фабрикой, нужен некто, кто будет наблюдать за конвейерной линией и проверять качество выходящих с нее продуктов. Этот некто должен бдеть неусыпно, иначе очень легко получить прекрасно автоматизированный процесс, дающий результаты обратительного качества. Для успеха процедуры сборки необходимо присутствие в цехе мастера; иногда его называют *царем сборки* или *бильдмейстером*.¹ Бильдмейстер следит за здоровьем сборочного процесса и обеспечивает устойчивый выход высококачественного продукта.

В заключение одно полутехническое замечание: остерегайтесь старых как мир заявлений типа «Мы всегда можем написать скрипт, который будет это делать». Проблема заключается в том, что подобные скрипты всегда начинаются с малого и простого, но потом быстро начинают развиваться случайно и бесконтрольно. В отличие от программ, скрипты очень редко пишутся по проекту. Они просто растут. Они мутируют в беспорядочный набор специальных случаев, абсолютно неадекватный постоянно растущим потребностям организации. Скрипты хрупки. Их очень трудно отлаживать. Сопровождение скриптов превращается в сущий кошмар, особенно если первый автор уходит из команды. Точно так же, как дорога в ад бывает выстлана благими намерениями, дорога в «сборочный ад» бывает вы-

¹ А вот поучительная, забавная и политически некорректная история: когда-то я придавал очень большое значение тому, чтобы у сборки был «царь», т. е. бильдмейстер, и назначил человека, скажем так, не высокорослого. И пожалел об этом, потому что называли его кто во что горазд, – кто царенком, а кто и вовсе цардинкой. Жуткое дело!

стлана вышедшими из-под контроля продуктами, написанными на языках сценариев общего назначения.¹

Как насчет итеративной разработки?

Итеративная разработка позволяет избежать одной из самых больших опасностей последовательного подхода – откладывания интеграции продукта до последней минуты. Одна из причин того, что такое множество последовательных проектов заканчивается крахом, заключается именно в том, что первые попытки собрать продукт происходят слишком поздно, в самом конце игры. Разработчики не только обнаруживают многочисленные дефекты, но и впервые сталкиваются с банальными организационными трудностями налаживания механизма сборки. Зачастую неполадки, которые выглядят как дефекты продукта, на самом деле оказываются не более чем результатами неисправности сборки. Но к этому моменту в организации царит такая неразбериха (времени не остается, ничего не работает, люди измотаны), что уже очень трудно отделить зерна от плевел. Кроме того, это очень неподходящее время для решения проблем политики и процесса.

В отличие от последовательного подхода, итеративная разработка требует создания механизма сборки с самого начала, иначе будет невозможно достичь цели первой же итерации – работающей программы. Поэтому отладка процесса сборки начинается на ранней стадии проекта, а не в его конце. Когда вы доберетесь до третьей или четвертой итерации, процесс сборки уже может идти довольно гладко. А во время последней итерации – той, которая призвана выдать на гора финальные биты, – сборочный процесс уже должен работать, как швейцарские часы.

Как и во всех остальных аспектах разработки ПО, при работе над механизмом сборки есть лишь несколько способов сделать все правильно и безграничное число способов сделать все не так. Если вы рассматриваете сборку как нечто, происходящее само собой, ваши шансы на успех невелики. Поэтому беритесь за процесс сборки с полным осознанием его важности и посвятите ему столько времени, усилий и ресурсов, сколько потребуется. Уделять ему меньше внимания – абсолютное безрассудство.

¹ Некоторые языки сценариев снисходительно сравнивают со скотчем (тем, который часто используют при монтаже воздуховодных труб). Представьте себе, что начинается заключительный этап сборки самолета, и какой-нибудь механик кричит: «Пора доставать скотч!».

Резюме

Меня часто вызывали на подмогу на поздних стадиях разработки ПО, когда проекты уже были в большой беде. Зачастую непросто с первого взгляда определить, насколько все запущено. Разработчики обычно больше всего беспокоятся о том, как сильно они «отстали», подсчитывая фрагменты, которые закодированы, но не работают, а также фрагменты, которые еще только предстоит закодировать.

Все это, конечно, важно, но я всегда изучаю и общее состояние процесса сборки. Если он «еле дышит» или отсутствует как таковой, необходимо *немедленное* вмешательство. Основание для такой тактики очевидно: на данной стадии развития отсутствие надежного, воспроизводимого процесса сборки будет препятствовать прогрессу во всех остальных отношениях. Невозможно тестировать то, что никак не собрать (и это в то самое время, когда постоянное и повторяемое тестирование является жизненно важным). Как же иначе разработчики будут знать, что они исправили, а что еще предстоит исправить?

Как это ни печально, во многих организациях процесс сборки спускают «игрокам второго состава». Это огромная ошибка. В этой части организации обязательно должны работать самые высококвалифицированные специалисты. Вы как высокопоставленный менеджер должны ясно показать, насколько высоко цените вклад тех, кто осуществляет процесс сборки. Как только люди это поймут, у вас не будет отбоя от добровольцев.

Есть еще один щекотливый вопрос – как определить, что этот этап завершен? Очень полезно заранее договориться относительно четкого критерия, чтобы не допустить резкого снижения или подъема планки с приближением даты поставки. Одна из важных задач при переходе в стадию передачи продукта при итеративной разработке – определение разумных критериев готовности к поставке. Без четкого «плана выхода» проект рискует попасть в череду бесконечных мелких исправлений в последнюю минуту.

На этом завершается часть II, посвященная основам. В части III я взгляну на разработку ПО с точки зрения управления проектами.

ЧАСТЬ ТРЕТЬЯ

Разработка ПО с точки зрения управления проектами

В части III разработка ПО рассматривается с точки зрения управления проектами. Она отражает приближение нулевого порядка, согласно которому управление проектом разработки ПО ничем не отличается от управления любым другим проектом.

В главе 9 «Компромиссы» я рассматриваю старинную проблему: «как засунуть эти восемь громадных помидоров в эту махонькую баночку?». Короче, почему всегда оказывается, что мы пытаемся слишком многое успеть в отведенное нам время?

В главе 10 «Оценка времени» я обращаюсь к некоторым элементарным понятиям оценки длительности осуществления программных проектов. Здесь снова появляется наш друг Роско Леруа.

В главе 11 «График работ» Роско продолжает излагать свою теорию планирования программных проектов, прибегая к некоторым методам научного прогнозирования.

Наконец в главе 12 «Ритм» мы увидим, что все успешные проекты и осуществившие их команды подчиняются некоторому ритму. Этот ритм обнаруживает поразительное единообразие в многочисленных проектах, поэтому я пытаюсь выяснить причины, которыми он вызывается.

Во всех этих главах я стараюсь сравнить и противопоставить проекты разработки ПО с «обычными» проектами и выявить как различия, так и сходства между ними.

ГЛАВА ДЕВЯТАЯ

Компромиссы

Самая большая трудность для каждого менеджера заключается в том, чтобы объяснить членам своей команды, что они должны успеть сделать всю необходимую работу к тому совершенно немыслимому сроку, который для них поставлен. Те самые разработчики, которые демонстрировали полный оптимизм («никаких проблем!») на подготовительных стадиях проекта, вдруг становятся мрачными и даже недовольными, когда руководство говорит им: «Ну, вперед!». Беда, конечно, в том, что руководство всегда хочет слишком многого. Хочет, чтобы все было сделано еще вчера.

Но это совершенно необходимо. Руководство знает, что если не ставить жестких сроков, то проект затянется еще сильнее. Беда в том, что большинству менеджеров программных разработок свойственна такая позиция: «Да, сроки напряженные, но команда хорошая, и, если повезет, мы справимся. Если мы примерно уложимся в сроки, все будет ОК». Напротив, разработчики думают по-другому. Они знают, что впереди их ждут многие неприятности: кто-то уйдет из команды, подрядчики, поставляющие важные компоненты, в одну прекрасную ночь сложат шатры и исчезнут в неизвестном направлении и т. д. Нарождающаяся система окажется слишком медленной в сравнении с прототипом. Проект просто захлебнется.

Руководство же рассматривает ситуацию как контракт: оно выделило средства корпорации для финансирования проекта и рассчитывает, что через сколько-то недель минута в минуту товар будет лежать на причале, готовый к погрузке в трюмы. Если выпуск задержится, начнется кошмар, потому что слишком много закручено разных шестеренок, для которых жизненно важен выпуск именно этого продукта.

Как возникает такое разобщение, и почему это случается снова и снова? Такой сценарий разыгрывается на моих глазах уже лет 40. Похоже, что мы никогда ничему не учимся. Простой факт таков: мы хотим сделать слишком много за слишком малый срок. Все прочие проблемы проистекают из этой фундаментальной ошибки.

Менеджер программных разработок – этот тот человек, который всегда оказывается между двух огней. Это он единолично поставил на «оптимистическую» оценку в графике работ, который из-за отсутствия резервов был обречен с самого начала. Он зажат между разработчиками, которые чувствуют, что их опять заставили идти путем камикадзе, и своим руководством, которое не в состоянии понять, почему никогда не удастся сделать работу вовремя. Иногда, пытаясь спасти свою шкуру, он соглашается выпустить продукт примерно в назначенный срок, последствия чего всегда одинаковы: наличие дефектов, недовольство клиентов, беспрерывные звонки в службу поддержки.

Есть две главные вещи, которые каждый менеджер программных разработок должен зарубить себе на носу. Во-первых, он должен *удерживать проект в начальных рамках*. Если вы не в состоянии управлять объемом того, что программируется, вы обречены погибнуть в удушающих объятиях «ползучего фичеризма». Во-вторых, это *примат времени*. Среди факторов, влияющих на окончательный исход, время оказывается тем, который управляет всеми остальными. В конце концов все остальное – это вопрос компромисса.

Пирамида проекта

Я давно нахожусь под обаянием ставшей теперь широко известной парадигмы «охват, ресурсы, время – выбери два пункта». Она утверждает, что попытка максимально расширить диапазон функциональности (охват)¹ при одновременной минимизации ресурсов и времени налагает чрезмерные ограничения и неизбежно приводит к провалу проекта. Макс Уайдман (Max Wideman) обсуждает этот «*железный треугольник*» на своем веб-сайте www.maxwideman.com, посвященном управлению проектами.²

¹ Термин «охват» (score) здесь и далее по тексту обозначает диапазон функциональных возможностей. – *Примеч. науч. ред.*

² Конкретные ссылки: <http://www.maxwideman.com/musings/irontriangle.htm> и <http://www.maxwideman.com/musings/triangles.htm>. В данной главе под *ресурсами* я понимаю все затраты, включая оплату труда.

Макс делает важное замечание, что в парадигму должно быть добавлено решающее четвертое измерение – качество. В своем письме ко мне он отметил:

Примечательно, что качество в конечном счете важнее всего остального, будь то интенсивность, продуктивность или окончательный продукт. Однако масса людей, занятых управлением проектами, этого, похоже, не понимают. Кому сегодня интересно, что прошлогодний проект не был выполнен в срок и превысил бюджет? Все это осталось в прошлогодних бухгалтерских документах. Зато качество [продукта] осталось.

Трудно не согласиться с такой точкой зрения. Большинство разработчиков ПО вспомнят, как им, старавшимся во что бы то ни стало выполнить свои обязательства и сдать продукт вовремя, случалось выпускать ПО такого качества, что потом они испытывали сильные сожаления.

Макс, таким образом, превращает треугольник в звезду (рис. 9.1).

Деррик Дэвис (Derrick Davis) в переписке с Максом предложил для иллюстрации этих связей заменить звезду тетраэдром. В результате сохраняется первоначальный треугольник, но добавляется третье измерение, отражающее аспект качества. Замечательным свойством тетраэдра является его внутренняя симметрия: четыре атрибута соответствуют вершинам, и любые три из них могут быть положены в основание. Макс умно проде-

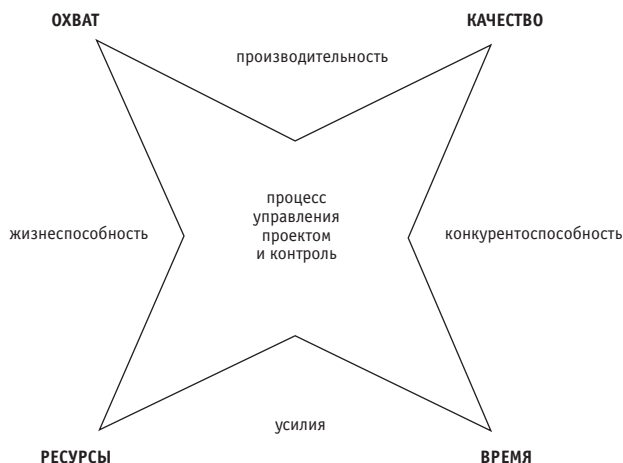


Рис. 9.1. Макс Уайдман развивает железный треугольник «ресурсы, охват и время», добавляя в него четвертый элемент – «качество»

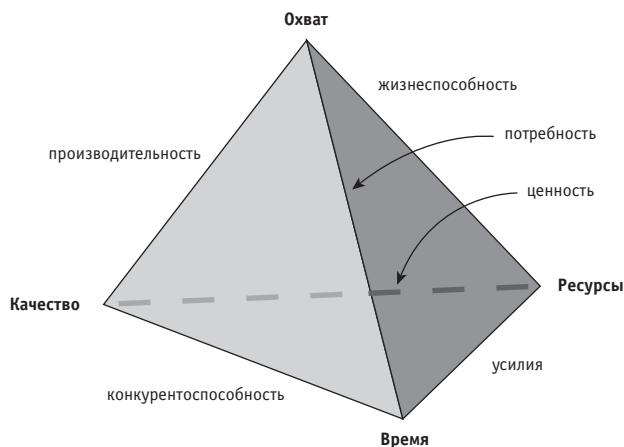


Рис. 9.2. Модель тетраэдра позволяет положить в основание любые три атрибута, а четвертый поместить в третье измерении

монстрировал это, связав пары вершин ребрами со смысловыми описателями (рис. 9.2).

Пять, а не четыре

Я согласен с Максом, что качество — это важный четвертый фактор, но все же считаю, что эта модель оставляет желать лучшего. Рассматривая проект перед началом работы над ним, руководство обычно интересуется его «видом», и этот интерес прекрасно соответствует четырем параметрам, показанным на рис. 9.1 и 9.2. Иными словами, мы можем определить, *что* мы собираемся сделать (охват), насколько *тщательно* (качество); мы можем предсказать, *когда* будет завершен проект (время), и оценить, *сколько это будет стоить* (ресурсы). Но заканчивается ли на этом описание проекта?

Думаю, что нет. Руководству всегда интересен пятый параметр — степень риска. То есть, исходя из выявленных ранее четырех параметров и сопутствующего плана, руководство желает знать степень риска, связанного с проектом, — высокую, среднюю или низкую. Наш богатый опыт подсказывает, что различные проекты характеризуются разными степенями риска, и хорошие менеджеры стремятся иметь портфель проектов, сбалансированный в отношении уровней риска. У более рискованных проектов больше вероятность провалиться, но они могут принести и более значительную прибыль. Так же, как рассудительный человек разнообразит свой

портфель ценных бумаг, умная компания диверсифицирует свой портфель, наполняя его проектами с разными сочетаниями риска/прибыли. Статистически такая компания должна быть финансово успешной.

Как же геометрически отразить этот новый (и, я надеюсь, последний) параметр?

Появляется пирамида

Я предлагаю модель, в которой первые четыре переменные соответствуют четырем ребрам в основании пирамиды. Мы припишем ребрам «*расширенные свойства*», чтобы длина этих сторон имела смысл. Обратите внимание на отличие от модели Дэвиса, в которой атрибуты соответствуют вершинам.

Допустим для простоты, что длины всех рёбер основания одинаковы, так что оно представляет собой квадрат. Похоже на звезду Макса, но атрибуты переместились от вершин к ребрам. Конечно, можно менять длину каждого ребра независимо, так что фактически основание является произвольным четырехугольником. Но в принципе ничего не изменится, если я пока предположу, что в основании лежит квадрат.

Теперь переопределим длину каждой стороны основания. Я также слегка изменю терминологию, чтобы она точнее отражала то, что представляют стороны. Потерпите, и вы поймете, почему я это сделал:

- **Охват.** Чем больше надо сделать, тем шире охват, поэтому длина этой стороны растет с расширением охвата.
- **Качество.** Более высокие стандарты качества требуют вложения большего труда, поэтому длина этой стороны растет вместе с ростом метрик качества – иными словами, когда мы «поднимаем планку» качества.
- **Скорость.** Это мера, отражающая время: длина этой стороны растет с увеличением скорости. И наоборот, чем медленнее вы двигаетесь (чем больше времени вам дано), тем короче становится эта сторона. Написать и отладить за месяц пять функций труднее, чем две. Эту сторону надо рассматривать как работу, выполняемую в единицу времени.
- **Экономия** (frugality). (Макс предложил это слово взамен предложенного мной сначала *parsimony* – *скупость*.) Если мы потребляем меньше ресурсов, то ведем себя более экономно. Чем сильнее экономия, тем длиннее эта сторона. Если мы расходует больше ресурсов, сторона укорачивается.

Обратите внимание, что при таком определении охвата, качества, скорости и экономии проект оказывается тем проще, чем короче стороны четырехугольника. То есть проект проще, если делать меньше, снизить требования к качеству, работать медленнее (потратить больше времени) и позволить себе не экономить (располагать большими ресурсами). Таким образом, все четыре переменные «направлены в одну сторону».

Заметьте также, что с ростом площади основания растет прибыльность. Действительно, ценность продукта увеличится, если он станет крупнее и лучше и будет готов быстрее при большей экономии потраченных на его создание средств. Максимизация ценности при минимизации издержек оптимизирует прибыльность. Совершенно логично, что, пытаясь увеличить свою прибыль, мы также делаем проект более сложным и рискованным.

Переменная высота

А теперь построим над этим основанием пирамиду, помня о том, что, каковы бы ни были размеры сторон для данного проекта, объем пирамиды пропорционален площади основания, помноженной на ее высоту. Высота служит абстрактным представлением *вероятности успеха проекта*, которая обратно пропорциональна величине его риска. Таким образом, у проекта с высокой степенью риска будет низкая вероятность успеха и маленькая высота. У проекта с низкой степенью риска будет высокая вероятность успеха и, соответственно, большая высота.

Теперь остается только связать все вместе (рис. 9.3).

Объем пирамиды представляет собой константу

Теперь мы можем утверждать, что объем пирамиды представляет собой постоянную величину для данной команды. То есть законы природы определяют, что в пирамиду проекта помещается лишь такое количество «вещества», какое соответствует способностям команды. Это и понятно, поскольку объем пирамиды пропорционален произведению:

$$\{\text{сложность}\} \times \{\text{вероятность успеха}\}$$

Когда увеличивается одно измерение, другое должно уменьшиться. Другими словами, здесь действует «закон сохранения»: произведение площади основания (она согласно определению четырех параметров представляет сложность проекта) на высоту (она представляет вероятность успеха) пропорционально «сохраняемому» объему.

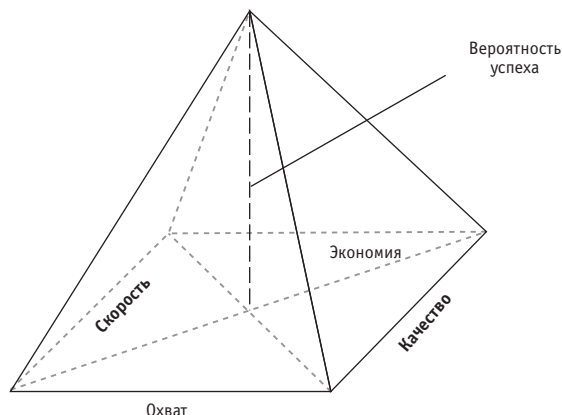


Рис. 9.3. *Пирамида проекта. У проекта с высокой степенью риска будет низкая вероятность успеха и маленькая высота. У проекта с низкой степенью риска будет высокая вероятность успеха и большая высота*

Чем определяется объем пирамиды? Двумя факторами. Во-первых, как я уже отметил, возможностями команды, работающей над проектом. Во-вторых, количеством незнакомых проблем, с которыми сталкиваются ее члены. Способной команде соответствует больший объем:

больше умения = больше «материала» = больше объем

Чем больше новых задач и неизвестных, тем меньше объем:

больше неизвестных = выше риск = меньше объем

Итак, что же надо сделать (при данном постоянном объеме, соответствующем команде проекта и начальному множеству переменных), чтобы увеличить высоту, т. е. вероятность успеха? По правилам элементарной стереометрии, надо уменьшить основание. Для этого достаточно сократить длину одной или более сторон основания, и тогда проект станет проще.

Запомните: объем пропорционален произведению площади основания на высоту независимо от формы этого основания.¹

Статистическая интермедия

Я сейчас попытаюсь определить правильный «масштаб» для высоты. Ребра, лежащие в основании, можно измерить в привычных единицах:

¹ Формула объема пирамиды: $V = (1/3) \times (\text{площадь основания}) \times (\text{высота})$.

- **Охват:** функциональные точки или основные функции.
- **Качество:** величина, обратная количеству дефектов.
- **Скорость:** функциональных точек или основных функций в месяц.
- **Экономичность:** величина, обратная потраченным долларам или человеко-месяцам.

А как быть с этой пресловутой вероятностью успеха – нашей высотой?

Мы знаем, что «чем длиннее, тем лучше»: большая высота соответствует большей вероятности успеха. Однако есть некоторые неувязки с использованием вероятностей (выражаемых в процентах от величины) в качестве шкалы. Например, если имеется пирамида с высотой, соответствующей вероятности успеха 60%, то нельзя, сохраняя постоянным объем, улучшить этот процент, сократив площадь основания вдвое. Это дало бы для вероятности успеха абсурдную величину в 120%, ведь значение вероятности должно находиться в интервале между 0% и 100%.

Чтобы справиться с этим затруднением, надо изучить распределение вероятностей исхода программного проекта. Можно ли считать, что исход проекта распределен согласно обычному нормальному закону, описываемому известной колоколообразной кривой? Один рис. 9.4 красноречивее тысячи слов.

Если вы позабыли, что такое функция распределения вероятности, напомним, что ось x представляет исходы, а ось y представляет относительное количество событий с таким исходом, которое при соответствующей нормализации (на общее число испытаний) становится вероятностью этого исхода. Если измерить площадь под кривой от левого ее края до данной точки, то она составит итоговую вероятность получения такого исхода. Проценты под осью x на рис. 9.4 показывают, какая площадь находится между концами охватывающих интервалов на оси x .

Обратите внимание, что распределение здесь «нормализовано» с центром в точке μ и «шириной» распределения, характеризуемой стандартным отклонением сигма (σ). Функция распределения уходит в плюс- и минус-бесконечность, но «хвосты» распределения за пределами плюс и минус ($3 \times \sigma$) весьма незначительны: оба они составляют менее 0,3% всей площади под кривой. График показывает, что 68% проектов оказываются в той или иной мере успешными или неудачными, что всего 27,5% (95,5% минус 68%) окажутся очень успешными или очень неудачными и лишь 4,2% (99,7% минус 95,5%) окажутся экстремально успешными или неудачными. Чтобы получить отдельный процент в каждом случае, надо поделить эти величины надвое, поскольку существует симметрия относительно центра.



Рис. 9.4. Связь исходов проектов программных разработок со стандартным нормальным распределением

Например, можно ожидать, что примерно 34%, т. е. примерно треть всех проектов окажутся в той или иной мере успешными.

Во многих приложениях принимается, что μ равно нулю, поэтому исходы лежат в диапазоне от минус бесконечности до плюс бесконечности. Но стандартное нормальное распределение применяется и в моделях, где исходы только положительные, например, если это человеческий рост. В таком случае μ сдвигается и представляет среднее значение распределения. Что можно сказать в этом смысле об исходах проектов?

Можно считать, что по оси x откладывается чистая денежная прибыль. Хотя я и не сомневаюсь, что у многих программных проектов прибыль нулевая или отрицательная, трудно представить себе, чтобы у какого-то проекта оказалась очень большая *отрицательная* прибыль.¹ Дело в том, что любой проект будет прекращен руководством задолго до того, как его доходность начнет приближаться к минус бесконечности! Поэтому симметричное стандартное нормальное распределение с серединой в нуле и концами, уходящими в бесконечность, представляется неверной моделью.

¹ Отчасти это связано с *конечным горизонтом* проекта. Если выпустить продукт с дефектами, то компания понесет большие затраты на последующую поддержку. Однако эти затраты редко относят на счет проекта. Это не совсем справедливо, поскольку снимает ответственность с тех, кто явился причиной убытков. По настоящему, следовало бы включать в стоимость проекта часть издержек на поддержку выпущенного продукта.

А что если «сместить» нормальное распределение? В этом случае также есть масса оснований предполагать, что распределение перекошено и имеет очень длинный (если не бесконечный) положительный хвост и более короткий отрицательный. Например, чистая прибыль такого проекта, как MS-DOS, огромна. Трудно представить себе проект, который не зарубили бы задолго до того, как его чистая прибыль станет симметричным отрицательным числом! Предпочтительнее выбрать распределение с конечной границей отрицательных исходов.

Идея правильная, распределение – нет

С этой целью мой дорогой друг и коллега Паскаль Леруа¹ предложил воспользоваться смещенным логнормальным распределением,² которое точнее отражает многие природные явления (рис. 9.5).

В отличие от стандартного нормального, логнормальное распределение асимметрично и у него нет левого хвоста, уходящего в бесконечность.

Стандартное отклонение по-прежнему обозначается буквой «сигма», но теперь мы интерпретируем его иначе, о чем будет сказано далее. Обратите внимание, что теперь μ совпадает с $\sigma = 1$. Половина площади под кривой находится слева от $\sigma = 1$, а другая половина – справа. Если принять, что все проекты описываются этим распределением, то нам нужно, чтобы наш проект оказался справа от линии 1 сигма, что будет означать превышение прибыли над средним значением.³ Это равносильно тому, чтобы сказать: мы готовы вложить в проект μ (или $\sigma = 1$), и любой меньший исход (полученное вознаграждение) будет означать убытки, а больший – прибыль.

¹ Никакой связи с Роско! Паскаль – реальный человек, который живет во Франции и часто помогает улучшить мои тексты своей вдумчивой критикой.

² Логнормальное распределение на самом деле образует целое семейство, из которого мы выбрали одного представителя. Подробнее о вездесущном логнормальном распределении можно прочесть в *LogNormal Distributions Across the Sciences: Keys and Clues*, Eckhard Limpert, Werner A. Stahl, and Markus Abbt, *BioScience* May 2001, Vol. 51, No. 5, page 341.

³ На самом деле с математикой тут немного сложнее. Для стандартного нормального распределения среднее и медиана одинаковы из-за симметричности распределения. В логнормальном распределении ее нет. В результате выбор 1 сигма не совсем точен, но на результате это почти не отразится. Мы можем игнорировать это расхождение, поскольку наша модель в целом не настолько точна, чтобы учитывать несколько процентов.

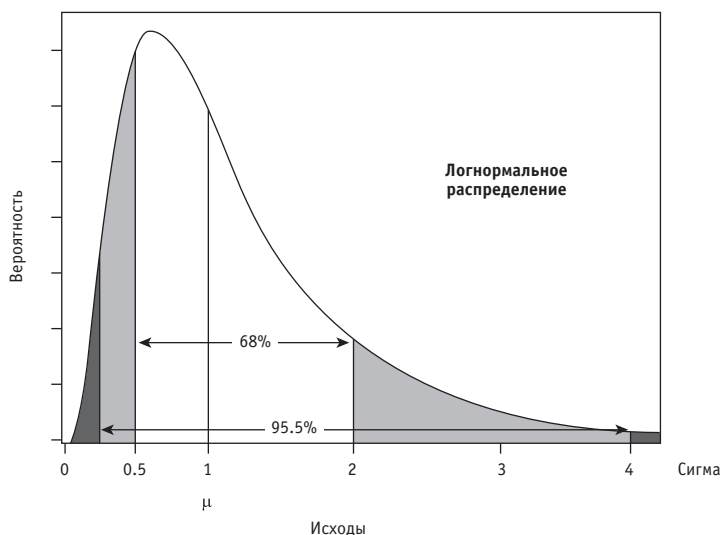


Рис. 9.5. Логнормальное распределение, отражающее только исходы с положительными значениями

Иными словами, мы «сместили» логнормальное распределение так, что μ ($\sigma = 1$) соответствует безубыточности, или нулевой чистой прибыли.

В отличие от стандартного нормального, логнормальное распределение помещает неудачные проекты между нулем и $\sigma = 1$, а успешные проекты — от $\sigma = 1$ до бесконечности, с длинным медленно убывающим хвостом. Это показывает, что справа находятся немногочисленные проекты с очень большой прибылью, но слева наши убытки ограничены. Представляется, что такая модель ближе к реальности.

Смысл сигмы в этом распределении иной. При движении от средней точки, обозначенной здесь как 1 сигма, площадь нарастает несколько иначе. Каждый доверительный интервал соответствует расстоянию от $((1/2)^n \times \text{сигма})$ слева до $(2^n \times \text{сигма})$ справа. Это означает, что 68% площади лежит между 0,5 сигма и 2 сигма и 95,5% площади лежит между 0,25 сигма и 4 сигма. Вот таким образом проявляется мультипликативная природа логнормального распределения.

Математически это распределение основано на событиях, статистически подчиняющихся *мультипликативной центральной предельной теореме*. Эта теорема показывает, как из многочисленных мелких случайных мультипликативных эффектов возникает логнормальное распределение.

В данном случае можно утверждать, что вся дисперсия исходов программных проектов обусловлена наличием множества мелких, но мультипликативных случайных эффектов. Напротив, стандартное нормальное распределение возникает при *аддитивном* вкладе множества мелких случайных эффектов.

Приложение к реальным проектам

Как приложить это распределение к реальным проектам? Поскольку пик кривой приходится примерно на 0,6 сигма, мы видим, что наиболее вероятным исходом (по меркам высоты кривой) оказывается провал проекта! Фактически, если бы пик приходился точно на 0,5, вероятность успеха составила бы лишь около 16%:

$$50\% - 1/2(68\%) = 50\% - 34\% = 16\%$$

Поскольку пик находится не в 0,5, а ближе к 0,6 сигма или 0,7 сигма, вероятность успеха несколько выше – примерно 20%.

Это очень любопытно, поскольку в отчете Standish Group CHAOS,¹ к которому я всегда относился несколько скептически, указана частота успеха, примерно равная 20%.² Я еще остановлюсь на этом отчете ниже. Но интересно отметить, что логнормальное распределение предсказывает величину Standish как наиболее *вероятный* исход, что может свидетельствовать о наличии в большинстве проектов разработки внутреннего фактора сложности, из-за которого возникает логнормальное распределение.³

¹ См, например, http://www.costxpert.com/resource_center/sdtimes.html

² Вот одна цитата из этого отчета: «... в целом разработка ПО заканчивается успехом в 24% случаев, причем для больших проектов эта цифра еще ниже, особенно в государственном секторе». В другом месте мы обнаруживаем замечание, что «...лишь 16% заявили, что регулярно успевают к назначенным срокам». Итак, есть две цифры: 16% согласно одному определению и 24% согласно другому. Поскольку я тяготею к крупным проектам, то можно снизить 24% до 20%, что подтверждает полученный результат. С учетом разницы индивидуальных представлений о том, что такое «успех», 20% или 25 малоразличимы. В любом случае такая вероятность успеха не радует.

³ И косвенным подтверждением мультипликативной природы влияния на результат факторов сложности, обсуждаемых автором. – *Примеч. науч. ред.*

Что надо, чтобы сыграть в «орлянку»?

Какой же менеджер захочет начинать проект с плохими шансами на успех? Хотелось бы иметь шансы хотя бы 50/50. Или, в терминах нашей модели пирамиды, что надо сделать с основанием, чтобы увеличить высоту?

Измеряя высоту нашей пирамиды в «сигмах», я начну с плана, который дает наиболее вероятный исход: в пике распределения при 0,66 сигма. Чтобы достичь вероятности успеха 50%, нужно набрать под кривой половину всей площади, что происходит в точке 1 сигма. Итак, мне надо дойти от 0,66 сигма до 1,0 сигма, что означает прирост в 50%. Таким образом, я должен увеличить высоту пирамиды в 1,5 раза, для чего площадь основания надо уменьшить в 1,5 раза, т. е. умножить ее на две трети.

Это в свою очередь означает, что длину сторон основания надо умножить на квадратный корень из двух третей, или примерно на 0,82. Поэтому, чтобы от наивного плана с 20% вероятности на успех перейти к плану с 50%, надо *одновременно*

- Уменьшить охват примерно на 18%;
- Снизить требования к качеству примерно на 18%;
- Продлить сроки примерно на 18% (иначе говоря, уменьшить скорость на 18%);
- Затратить примерно на 18% больше ресурсов (другими словами, снизить экономичность на 18%), чем было запланировано в первоначальном сценарии;

Можно, конечно, изменить эти параметры и в других пропорциях, лишь бы площадь основания уменьшилась на треть.

Назовем этот новый план – тот, который приводит нас к результату 50/50, – «планом В». Первоначальный же, наиболее вероятный и наивный план, назовем «планом А».

Рост уверенности

Есть ли варианты получше? Допустим, я хочу добраться до точки 2 сигма. Это будет означать вероятность успеха, близкую к 84%:

$$50\% + S(68\%) = 50\% + 34\% = 84\%$$

В результате наши шансы станут равны пяти к одному, на что с радостью согласится любой менеджер. Действительно, отчет Standish будет поспрашен: пять успешных проектов на каждый неудачный.

Во что нам это обойдется?

Можно вести расчеты двумя способами: опираясь на план А или на план В с шансами 50/50. Для единообразия возьмем план А. Арифметика примерно та же. От 0,66 сигма теперь надо перейти к 2 сигма, увеличив высоту втрое. Это означает, что требуется умножить площадь основания на треть, что в свою очередь означает умножение каждой стороны основания на квадратный корень из 0,333. Теперь для получения нужного результата в прежнем списке параметров, которые должны быть одновременно изменены, надо на место 18% поставить 42%.

Подведем теперь итоги, пользуясь округленными числами, чтобы не создавать иллюзии точности нашей модели.¹ Вероятность успеха по плану А составляет лишь около 20%. Мы видели, что если уменьшить сложность каждого из четырех основных параметров (охват, качество, скорость и экономичность) на 20%, то получится план В, в случае чего вероятность успеха составляет 50%. Чтобы достичь вероятности успеха в 85%, надо уменьшить сложность по этим базовым параметрам примерно на 40% по сравнению с планом А. Эти соотношения подытожены в табл. 9.1.

Таблица 9.1. Результаты применения пирамидальной модели и логнормального распределения

План	Описание	Положение на кри- вой логнормально- го распределения	Вероятность успеха	Значения базовых параметров
А	Наивная и наиболее вероятная отправная точка	0,67 сигма	20%	Согласно плану
В	Более реалистичная	1 сигма	50%	На 20% меньше, чем в А
С	Высокоэффективная	2 сигма	85%	На 40% меньше, чем в А

Очевидно, что мы скользим здесь по очень тонкому льду, но цифры в табл. 9.1 дают предсказания модели, основанной на пирамиде, если счи-

¹ Вспомним также проигнорированную нами раньше деталь, касающуюся несовпадения среднего и медианы для логнормального распределения. Здесь мы можем частично отказаться от того приближения.

тать, что имеет место логнормальное распределение исходов проектов, а объем не меняется.

Важные предостережения

Теперь необходимо вернуться немного назад и рассмотреть ограничения данной модели. Я сделал много неявных предположений и теперь хочу их раскрыть.

- Начнем с того, что подразумевается под словом «успех». Если помните, то я сказал, что определяю успех как исход, превышающий 1 сигма в логнормальном распределении, из чего следует, что примерно половина наших проектов окажется успешной.

Однако в отчете Standish говорится, что четыре из пяти проектов проваливаются. Не значит ли этот результат, что к ним предъявляются очень жесткие требования и потому они столь сложны? Возможно. Многие программные проекты обречены уже в тот момент, как высохли чернила, которыми написан их план. Но мне кажется, что дело не только в этом.

Отчет Standish всегда вызывал у меня сомнения, поскольку, по моему мнению, он содержит преувеличения, а потому упрощает реальную проблему. Если взять все первоначальные планы проектов и оценить их в момент завершения по нашим четырем базовым параметрам, то Standish, возможно, окажется права. И логнормальное распределение согласуется с таким сценарием. Но неужели у нас действительно на каждый успех приходится четыре провала?

Мне кажется, на самом деле происходит вот что: по ходу реализации проекта руководство приходит к выводу, что изначально оно выдвинуло слишком честолюбивые цели, или что разработчики были чрезмерно оптимистичны, или что разработчики не вполне понимали поставленную задачу. Но на проект уже потрачены деньги, и выбросить его на помойку было бы расточительно и неразумно. Поэтому проект исправляют и ставят для него новые цели. При этом может быть сокращен набор реализуемых функций или часть из них отложена для включения в следующие версии. Иногда, особенно если команда сталкивается с трудностями уже в конце жизненного цикла, она жертвует качеством и выпускает продукт с многочисленными дефектами. И даже в этом случае команда может выйти

за рамки графика и бюджета. Но значит ли это, что проект провалился? Необязательно.

Я утверждаю, что многие проекты попадают в класс «достаточно успешных» или в класс «частично успешных». Как это бывает в жизни, мы по ходу дела пересматриваем свои ожидания (обычно в меньшую сторону), чтобы объявить о своей победе, когда они сбудутся. Это необходимо как с политической, так и с психологической точки зрения. Таким образом мы избегаем того, что психологи называют *внутренним конфликтом*. Никто не любит поражения, и всегда можно что-то спасти. Поэтому мы стремимся незаметно подправить историю и смошенничать с результатами. Цифры Standish справедливы только в том случае, если оценка основывается на исходном проекте. Но на практике никто так не поступает. В таком контексте правильной оценкой будет признание успешными примерно половины проектов.

- Такие параметры, как охват, качество, скорость и экономичность, не являются взаимно независимыми. Например, если проект затягивается по срокам, то его стоимость увеличивается из-за возросшего расхода ресурсов, поэтому скорость и экономичность страдают одновременно. Можно попытаться поправить одно с помощью другого, например, потратить деньги и нанять еще людей, чтобы двигаться быстрее. Но как очень верно заметил Брукс почти 30 лет назад,¹ добавление в программный проект новых работников² обычно приводит к его замедлению! Если вас интересуют подобные компромиссы, то лучше вопреки здравому смыслу расходовать меньше средств в единицу времени, сократив количество работников и продвигаясь медленнее. Возможно, вы даже не сильно проиграете во времени, поскольку, как отметил Брукс, меньшие команды оказываются более эффективными.
- Различные параметры неравноценны по своему значению. Время – величина, обратная скорости, – играет более важную роль, чем все остальные, хотя об этом можно и поспорить. Обычно менеджеры сопротивляются сокращению охвата и снижению качества и при этом всегда очень торопятся. С их точки зрения, единственным па-

¹ Фредерик Брукс «Мифический человек-месяц или как создаются программные системы». – Пер. с англ. – СПб.: Символ-Плюс, 2000.

² В ходе уже выполняющегося или приближающегося к завершению проекта.

раметром, который они могут варьировать, являются ресурсы. Поэтому они часто пытаются решать проблему, изыскивая дополнительные денежные средства. Обычно это ничего не дает, поскольку они не дают себе труда подумать, как распределить эти средства, и тратят их неэффективно. Это полностью совпадает с точкой зрения Брукса. Он говорит, что в конечном итоге больше проектов проваливается из-за недостатка времени, чем по всем остальным причинам вместе взятым. Он был прав тогда, и я считаю, что он прав до сих пор.

- Обычно не удастся компенсировать один из этих четырех параметров за счет другого – во всяком случае в больших размерах. Имеется в виду, что большой дефицит в одном из них нельзя восполнить путем значительного приращения одного или нескольких других. Проектам свойственно подчиняться закону естественного равновесия: если вы попытаетесь выстроить основание, в котором одна из сторон будет значительно отличаться по размеру от остальных, вас ждет неудача. По этой причине я сделал предположение, что основанием у нас является квадрат и все стороны (параметры) примерно одинаковы. Я допускаю, что можно увеличивать или уменьшать стороны, чтобы получить искомую площадь, но имейте в виду, что при этом необходимо соблюдать меру. Нельзя произвольно менять один параметр, чтобы обойти препятствия, связанные с другими. Макс Уайдман любит уподоблять основание «листу резины». Можно тянуть его за угол, изменяя длины сторон, но если переусердствовать, он порвется. Конечно, с точки зрения геометрии одна сторона не может быть длиннее суммы длин трех других, иначе четырехугольник «не замкнется».
- В известной мере я проигнорировал самый важный фактор любого проекта разработки ПО: талант участвующих в нем людей. Я неоднократно убеждался, что значение имеет не просто количество людей, работающих в команде, но их мастерство, опыт, пригодность и характер. Темы управления динамикой команды и подбора специалистов для конкретных задач проекта выходят за рамки данной главы. Однако объем пирамиды в некоторой степени соответствует одаренности команды.
- Не следует определять качество продукта только по количеству имеющихся в нем дефектов. Качество лучше определить более общим образом, таким как «пригодность к применению». Если продукт не со-

держит ошибок, но не может удовлетворительно решить важную задачу, то в целом он бесполезен и не может быть признан «высококачественным».

- А как с итеративной разработкой? К сожалению, подход, рассматриваемый в этой главе, трактует проект как «одноразовое действие», что противоречит всем принципам итеративной разработки. Не исключено, что необычайно высокий процент провалов, отмеченный Standish, как раз вызван *отсутствием* итеративной разработки. Иными словами, начав с нереалистичного плана и твердо придерживаясь его на протяжении всего проекта, несмотря на получение данных, свидетельствующих о необходимости корректировки, мы и приближаем собственный провал.

Однако, если у нас хватит ума, чтобы воспользоваться итеративным подходом, можно предложить некую работоспособную модель. На этапе исследования начнем с пирамиды определенного объема и высоты, исходя из того, что в тот момент нам известно о команде и неизвестных параметрах. При переходе к следующему этапу пирамида может изменить как свой объем, так и форму. Объем может измениться в результате повышения или снижения возможностей команды или благодаря новым знаниям, позволяющим уменьшить риск. Это естественное следствие итеративной разработки. Кроме того, может измениться форма пирамиды, если мы установим новую длину для одной или нескольких сторон основания путем некоторого сокращения области, добавления ресурсов, увеличения срока или послабления в отношении качества или путем действий в противоположном направлении. Это должно происходить на границе каждого этапа, и всякий раз нашей целью должно быть увеличение высоты. По мере прохождения нашего проекта через все четыре фазы итеративной разработки мы должны наблюдать не только рост объема пирамиды, но и последовательный рост ее высоты, поскольку мы всеми силами стремимся снизить риск. Если не получается сделать это путем увеличения объема, надо добиться того же с помощью уменьшения площади основания.

- Вопрос о том, описываются ли проекты логнормальным распределением вероятности, является спорным. Я согласен с Паскалем в том, что это распределение предпочтительнее стандартного нормального. И вот почему.

Нормальное распределение получается при сложении большого числа мелких событий, влияющих на окончательный результат. В разговорной речи мы можем сказать: «Что-то окажется лучше, чем мы предполагали, что-то хуже, но в итоге все усреднится». Здесь неявно участвует понятие симметрии, т. е. представление о том, что исход может с равной вероятностью оказаться больше или меньше своего среднего значения. Эти два допущения – сложение и симметрия – делают нормальное распределение симметричным относительно центрального значения и уводят хвосты в бесконечность в обоих направлениях, т. е. вероятность того, что все события будут иметь только одно или другое направление, мала. Если результат образуется суммированием «симметричных» исходов, то нормальное распределение оказывается очень надежной концепцией; в действительности так называемая *центральная предельная теорема* фактически гарантирует его получение, даже если сами суммируемые величины в отдельности не подчиняются нормальному распределению.

Нормальное распределение, или «колоколообразная кривая», вошло в наше коллективное сознание. Оно является главной темой всех читаемых курсов теории вероятности и статистики, оно интуитивно и обладает рядом легко вычисляемых характеристик. Но в жизни не всегда результат получается путем сложения образующих его величин. Поэтому следует с осторожностью применять стандартное нормальное распределение, если не изучены лежащие в основе явления причины. Несмотря на поразительно широкую область его применения, будет ошибкой предполагать, что «стандартное нормальное» действует всюду.

Логнормальное распределение имеет место, когда многочисленные мелкие величины, порождающие конечный результат, перемножаются. Обратите внимание, что между сложением множества маленьких величин и их перемножением есть существенная разница. Для получения большого результата при перемножении необходимо, чтобы некоторые сомножители были «большими», но для того, чтобы результат был близок к нулю, достаточно только одному из сомножителей быть близким к нулю. Поэтому распределение несимметрично и смещено в сторону малых величин. А ведь правда, многие составные события в жизни лучше моделируются логнормальным распределением: чтобы получился большой положительный результат, очень многое должно состояться правильно; чтобы получился ма-

ленький результат (так сказать, негативный исход), достаточно лишь одной очень неправильной вещи. Даже один «нуль» обнулит результат, какими бы ни были все остальные множители. В случае управления проектами опыт подсказывает мне, что природа распределяет исходы скорее логарифмически нормально, чем просто нормально.

- Наконец, закон сохранения, выражаемый как постоянство объема пирамиды, представляет собой лишь модель. Она обеспечивает удобную визуализацию явления, но это всего лишь предположение, простейшая геометрическая модель, какую я смог придумать. Чтобы установить, отражает ли она реальность, необходимо изучить эмпирические данные.

Список предостережений получился длинный, но он не отрицает ценности модели. Я считаю, что ее предсказания обоснованны и согласуются с моим прежним опытом. На самом деле множественные корректировки курса, осуществляемые командами во время работы над проектом для повышения вероятности его успеха, оказываются лишь паллиативами, не решающими реальных проблем. В нашей профессии было неоднократно продемонстрировано, что существенного увеличения шансов на успех не добиться путем простого ослабления ограничений на 10%, и данная модель подчеркивает это обстоятельство. В этом и заключается ее главная ценность; я полагаю, что она представляет фундаментальную истину.

Все дело в риске

Риск, по-видимому, является важнейшим параметром, который надо учитывать при финансировании и планировании проекта. Вот почему простая модель, которую я описал в этой главе, связывает четыре традиционных параметра проекта – охват, качество, скорость, экономичность – и добавляет риск в качестве пятой переменной. Если вы собираетесь спускаться по реке на каноэ, желательно знать, к какому классу относятся пороги – к третьему или к пятому. В последнем случае риск значительно возрастает, и, может быть, стоит потратить больше денег на лодку. То же происходит с инвестициями в разработку ПО: стоит оценить риск, прежде чем решать, какие ресурсы надо выделить для проекта. Но помните, что ресурсы представляют собой лишь одно ребро в квадратном основании; их следует рассматривать согласованно с охватом, временем и качеством. Именно комбинация этих параметров наряду с качеством команды в конечном счете определяет уровень риска.

Данная простая модель пирамиды также показывает, какие уступки вам *придется* сделать, чтобы повысить вероятность успеха. Несмотря на свою абстрактность, эта модель позволяет трезво оценить, готовы ли мы обеспечить ресурсы, необходимые, чтобы поднять вероятность успеха выше минимального порога, приемлемого в нашем бизнесе, исходя из заданных ограничений на охват, качество и время.¹

Резюме

Нетрудно представить себе, что я подвергнусь острой критике за свои математические модели. Причин тому несколько.

Прежде всего существует интересная культурная традиция. Наша цивилизация за прошедшие тысячелетия потратила несметные деньги на создание современного математического аппарата, а средний менеджер оперирует какими-то жалкими крохами этих сокровищ. Непостижимо. Ведь мы же учим школьников математике! Обычно говорят, что «данные неправильные» или «данных нет», но мне кажется, что это просто отговорка.

Почему мне нравятся математические модели? Мой ответ прост. Все мы примерно понимаем или инстинктивно чувствуем, как устроен реальный мир. А вот что мы не понимаем, так это размеры и тяжесть последствий своих действий. Если менеджер проекта, оказавшегося в трудном положении, руководствуется здоровыми инстинктами и, например, решает уменьшить охват, а не брать новых людей, то это движение в верном направлении. Но было бы еще лучше, если бы он знал, *насколько* надо уменьшить охват, чтобы исправить положение. Если таких данных нет, то менеджер скорее пойдет лишь на незначительное уменьшение охвата. Так он навлекает на себя гнев руководства, во-первых, за то, что проект выполнен лишь частично, и, во-вторых, за то, что он все равно опоздал, т. к. выкинул мало. Из двух зол выбраны оба.

Предлагаемые мной модели просты по замыслу. Обычно они требуют знания математики в пределах понятия площади и объема и закона сохранения (если что-то где-то прибудет, то в другом месте это что-то должно

¹ С моего одобрения Макс Уайдман включил часть материала этой главы в свою новую книгу «A Management Framework for Project, Program and Portfolio Integration» (Организационная структура для интеграции проекта, программы и портфеля) (Victoria: Trafford Publishing, 2004). Глава 8 этой книги начинается с пирамиды проекта и развивает эту идею. Поскольку я начал собственное исследование с «его» тетраэдра, это совершенно правильно и уместно.

убыть). В данной главе я обратился к некоторым понятиям статистики, но постарался тут же их объяснить. Логнормальное распределение – вещь неочевидная, но это не умаляет его пригодности. Это правильно выбранное распределение.

Конечно, очень трудно сравнить результаты этих моделей с экспериментальными данными. Но я не думаю, что это должно удерживать от создания моделей. Если вам не нравится моя, придумайте свою собственную. Посмотрите, насколько ваши результаты будут отличаться от моих. Я неоднократно убеждался в том, что когда появляются три или более конкурирующие модели, их предсказания оказываются удивительно близкими. То есть, если модели правильно отражают реальность в каком-то отношении, все они дадут аналогичные результаты независимо от деталей.

Модель, построенная на компромиссах, учит нас, что повысить вероятность успеха не так-то просто. Необходимо существенно изменить параметры, чтобы добиться заметного результата. В свою очередь это означает, что наши усилия по выработке оценок и графиков работы должна быть более эффективной. Я рассмотрю эти темы более подробно в главах 10–11. Как можно догадаться, у Роско Леруа найдется что сказать по этим вопросам. Будет интересно сравнить некоторые его математические модели с моими.

ГЛАВА ДЕСЯТАЯ

Оценка времени



Одна из загадок программирования состоит в том, что неизвестно, почему даже лучшие программисты очень плохо оценивают время, необходимое им для выполнения работы. Их оценки отличаются либо неумеренным оптимизмом, либо, напротив, крайним консерватизмом. Они могут сказать, что им потребуется «пара недель», чтобы создать некоторую крупную подсистему с нуля, и *в тот же день* назвать такой же срок для работы, в которой надо исправить какие-нибудь две строчки кода, что свидетельствует о какой-то аномалии. Еще хуже, когда две такие оценки исходят от одного и того же программиста, но это случается довольно редко!

Вынужден признаться, что мне не удалось понять, как добиться точности в оценке времени. Я лишь обнаружил, что молодые и менее опытные разработчики склонны к излишнему оптимизму. С этим можно бороться, подвергая их оценки критике со стороны старших людей, включая архитектора проекта. Иногда разработчик предполагает применить в реализации особую «хитрость», которая позволит очень быстро решить задачу. Но этот расчет будет ошибочным, если окажется, что его подход не стыкуется с архитектурой проекта в целом. Поэтому такие аномальные оценки следует рассматривать самым внимательным образом и тщательно обсуждать. Обычно в результате удастся лучше понять проблему и точнее оценить срок выполнения, хотя он и бывает увеличенным.

А что предсказатели из другого лагеря? Чрезмерно консервативные оценки возникают, когда проблема детально изучается, начиная с самого низкого уровня, и на каждом шаге предусматриваются меры безопасности.

Все эти меры безопасности накапливаются, как будто вы предполагаете, что все неприятности ждут вас на каждом шагу, что случается редко (вопреки закону Мерфи). Эти оценки также подвержены *эффекту обособления* (*compartmentalization effect*). Хорошие разработчики трудятся сразу над группой проблем, особенно при исправлении программных ошибок. Например, часто несколько проблем могут быть связаны с каким-то одним фрагментом кода, который необходимо переработать. Наведя порядок в этом разделе кода, можно избавиться сразу от нескольких проблем. Если оценивающий исходит из того, что все ошибки будут исправляться порознь, то его результат может в два-три раза превышать реально необходимое время.

Некоторые разработчики строят для себя систему защиты. То есть они умышленно завышают оценки, потому что «обожглись» на сроках в прошлом или не очень хотят заниматься данной задачей. Хороший менеджер должен быть очень терпелив и работать над тем, чтобы установить атмосферу доверия в отношении оценок. Никого нельзя строго наказывать за то, что он не смог дать объективную оценку и не скрыл это. А вот с безответственностью оценок (определением сроков без достаточных оснований) и перестраховкой надо активно бороться.

Не прибегнуть ли к здравому смыслу?

Иногда мы слишком увлекаемся профессиональным жаргоном: структура процесса, множественное наследование, полиморфизм и т.д. А как быть, если надо объяснить эти понятия «рядовым гражданам»? Как они воспримут наши методы оценивания? Роско Леруа высказывает свою точку зрения на оценку сроков.¹

Шоколадное или ванильное?

Однажды Роско появился у меня в доме с номером «USA Today» под мышкой. «Похоже, что даже McPaper² начинает что-то понимать», – сказал он, зная, что я не премину заглотить наживку. Когда я поинтересовался, о чем он говорит, он показал мне короткую заметку в углу страницы: 52% населения предпочитает ванильное мороженое, а 42% – шоколадное.

¹ Если вы не читали главу 5 «Самое важное», то не знаете, кто такой Роско. Можете прочесть сейчас, как он там представлен, чтобы лучше понять, что это за человек – Роско.

² Неофициальное пренебрежительное название газеты «USA Today».

С моей точки зрения, это сообщение соответствовало довольно высокой метке по шкале «ну и что?».

Роско продолжил.

—Обрати внимание, что написано под столбчатой диаграммой: достоверность (ассигасу) оценки составляет плюс-минус 3%.¹ Это важно. Они годами печатали такие вещи, ничего не сообщая о величине ошибки. Должно быть, мы стали больше понимать в опросах, если они считают, что нам надо сообщить, какова возможная ошибка».

Я вынужден был признаться, что никогда об этом не думал. Меня также заинтриговало, почему Роско проявил к этому интерес.

Разъяснения Роско

— Три процента ошибки, — сказал он, — это немало, потому что разрыв между предпочтениями составляет всего 4%. Поэтому полной уверенности быть не может. Но я тебе вот что скажу: бьюсь об заклад, что знаю, сколько людей они опросили.

Я был уверен, что Роско никогда не участвовал ни в каких «семинарах» по теории вероятностей и статистике, где не фигурировали бы фишки для покера, но, похоже, я ошибался.

— Вот что я думаю, — продолжил он. — Они опросили примерно 1000 человек. Ошибка для 1000 ответов составит примерно корень квадратный из 1000, т. е. 31,4. Тридцать один на 1000 составит 3,1%, т. е. примерно 3%.

— Подожди, Роско, — не удержался я, — квадратный корень из 1000 равен 31,6, а не 31,4.

¹ Иногда употребляется термин *precision* (точность), но достоверность (или доверительность) и точность — это не одно и то же. Точность указывает на возможность воспроизвести измерение. Если есть метод высокоточного измерения, с помощью которого многократно выполняется замер, то большинство результатов попадет в узкий интервал. Напротив, достоверность (*assigasu*) служит мерой того, насколько близок результат замера к реальности. Точность относится к проведению измерений на выборке; достоверность говорит о том, насколько измерение, сделанное по выборке, приближается к свойствам всей совокупности, в данном случае населения.

Если вам показалось, что Роско не слишком аккуратно обращается с этими терминами, то вы не правы. Нам тоже следует их различать. Гнаться за высокой точностью бессмысленно, если достоверность невысока: высокая точность стоит дорого, а зачем она нужна, если результат неправильный! Помните, что высокая точность не всегда означает высокую достоверность.

– Опять ты лезешь со своим образованием. Любой дурак может запомнить, что квадратный корень из 10 – это число «пи», т. е. 3,14. А для 1000 надо переместить в ответе на один знак десятичную точку – вот тебе и 31,4.

Моя арифметика была точнее, но с учетом последующего округления разница была не принципиальна. И эти расчеты производили впечатление вполне обоснованных.

Роско идет дальше

– И еще скажу, – продолжал Роско. – Ты никогда не увидишь заметки, в которой говорилось бы, что погрешность составляет, скажем, 1%.

Пока я несколько секунд раздумывал над его словами, Роско потерял терпение.

– Не нужно быть Эйнштейном, чтобы понять почему. По той же логике для снижения ошибки до 1% надо опросить *10 000 человек*. Это в 10 раз дороже, чем опросить тысячу. Поэтому, чтобы понизить ошибку с 3% до 1%, надо увеличить издержки в 9 раз.

– В десять, – подумал я. Потом понял, что первую тысячу опрошенных можно учесть и во второй выборке. Да, этого Роско не проведешь.

Календарь Роско

К этому времени я уже поверил, что Роско знаком с квадратными корнями. Это подтвердилось, когда я заговорил с ним о проектах и графиках работы. Его представление о том, сколько недель содержит тот или иной отрезок времени было, мягко говоря, любопытным. Мое открытие сведено в табл. 10.1.

Таблица 10.1. Календарь Роско

Продолжительность	Продолжительность по Роско
Два года	100 недель
Один год	49 недель
Девять месяцев	36 недель
Шесть месяцев	25 недель
Четыре месяца	16 недель
Два месяца	9 недель
Один месяц	4 недели

Когда я поинтересовался трехмесячными проектами, Роско ответил: «Не люблю ими заниматься».

Эти числа ясно показывали, что тут действует какая-то очередная особая арифметика Роско. Такие схемы не устанавливают без определенных на то оснований. Любопытно также было видеть, что Роско не желает иметь дело с промежутками времени короче недели. «Мы, сынок, не поденщики, – заявил он. – Я плачу своим инженерам понедельно. Потому я не пользуюсь для оценок меньшими интервалами».

Столь же логично, как его рассуждение о дробном количестве работников.

Роско вычисляет

– На самом деле, – сказал Роско, – для меня вычислить квадратный корень раз плюнуть. Без всякой кнопки «кв. корень» на старом калькуляторе.

– Пятьдесят три! – крикнул я, тут же бросив ему вызов.

– Запросто, – сказал он, болтая и расправляясь с числами одновременно. – Пятьдесят три близко к 49, поэтому начнем с семи. В 53 семь умещается 7,57 раз. Ясно, что 7 слишком мало, а 7,57 слишком много, поэтому выберем среднее. Сдается мне, что это 7,28. А 7,28 помещается в 53 – ты просто не поверишь – 7,28 раза. Так что я полагаю, 7,28 – это правильный ответ.

Извлечь квадратный корень с двумя знаками после запятой за два деления – это впечатляет. Но у Роско иная точка зрения.

– Слишком утомительно, – заявил он. – Предпочитаю работать с круглыми числами. Ну, там, с динамитными пашками или целыми людьми...

Роско обращается к программированию

Только я подумал, что мир в безопасности, как Роско сделал новое признание. «Шахту закрыли, сынок, – сказал он. – Похоже, нужно искать новую работу. По-моему, я бы справился с каким-нибудь из этих программных проектов».

«Слава богу, что «дот-комовское» безумие утихло», – подумал я. Разве я сумел бы объяснить ему *это*? С другой стороны, он, несомненно, привнес бы свежий взгляд в наш бизнес. Кое-чему у него можно было бы поучиться.

«Конечно, Роско, – отвечал я. – Займись изучением итеративной разработки. Лучше всего начать с этого. Почитай что-нибудь на эту тему, поговори с людьми и возвращайся». Я надеялся, что это займет его на какое-то время.

– Я к тебе еще вернусь, – сказал он, уходя и стараясь не хлопнуть наружной дверью.

Роско докладывает

Примерно через месяц Роско появился, как обычно, без предупреждения и готовый к драке. Я прикинул, что ему должно было хватить времени, чтобы разобраться с основами, и мне было интересно услышать, что он скажет.

«Похоже, понаписано об этой штуке немало, – высказался Роско, подтягивая к себе трехногий табурет. – Просто поразительно, как можно столько написать о таком запутанном предмете».

Я признал, что процент успеха наших программных проектов оставляет желать лучшего. На Роско это не произвело впечатления.

– Похоже, ребята считают, что это очень тяжелое занятие. Чушь! Вот уголь добывать – это действительно тяжело. А здесь... Ведь не с железом работают!

Я сказал, что, может быть, слово «трудный» более подходит, чем «тяжелый», но Роско был неумолим.

– Позволь тебе напомнить, что пирамиды построили египтяне. Правда, у них был этот Евклид, совсем не дурак, что и говорить.¹ Но аманиты и сегодня могут соорудить коровник за день и не станут поднимать из-за этого шум. Хотя, – продолжил он, – некоторые обнадеживающие признаки есть.

Кое-что мы делаем правильно

– Мне кажется, что эти ребята на правильном пути со своей итеративной разработкой. Хотя это просто красивое название для того, чем занят каждый сельский плотник – «отпили и приложи».² Они знают, что с нуж-

¹ Хороший пример того, что полученное Роско образование ему не помеха: Евклид появился на свет позднее, чем пирамиды, и намного. Он жил около 300 года до н. э., а пирамиды построили около 2500 до н. э. Что делает пирамиды еще более впечатляющими, не правда ли?

² Леруа говорит здесь о плотницких работах, которыми занимаются фермеры и скотоводы, – постройкой грубых сооружений типа сарая или забора. На другом конце спектра находится тонкое ремесло, которым занимаются столяры или краснодеревщики. Я думаю, что большинство программистов-профессионалов считают себя скорее краснодеревщиками, поэтому Роско провел не совсем удачную аналогию. Но есть и промежуточная область. Мы же не начинаем программные проекты по подробному чертежу с точностью до миллиметра.

ной точностью все равно не отмеришь и не отпилишь, поэтому сначала делают навскидку, а потом подгоняют по месту. Но вы, конечно, назовете это «постепенным уточнением при последовательных итерациях».

Я заметил нотку сарказма в сделанном Роско заявлении. Но его было не удержать.

– Конечно, плотнику нужно быть внимательнее, чем вы себе позволяете. Он знает, что всегда можно сострогать еще немного, а вот нарастить обратно будет посложнее. Научился этому у своего парикмахера. Поэтому первый раз всегда нужно отпилить правильно. В больших городах контролеры тоже иногда пользуются этим алгоритмом.

Слово «контролер» он произнес правильно и, надо полагать, понимал его. Но «алгоритм» подозрительно походил на «алко-ритмы». «О, черт, – подумал я, – у нас большие неприятности: Роско выучил новое слово».

– Роско, где ты подцепил это словечко – «алгоритм»? – спросил я.

– Сынок, я тут поговорил с некоторыми программными архитекторами... – начал было он.

– Так, – сказал я, – это кто же тебе посоветовал?

– Прежде всего ты сам. Ты же сказал мне – поговорить с людьми. Я могу показаться дураком, но я знаю, чего я не знаю. Есть большая разница между чтением чертежей и умением делать их самому. Так что я пошел и немного поболтал с этими славными ребятами.

– Согласен, – сказал я. – И что же ты выяснил?

Роско подытоживает

– Мне все понятно. Обычно, начиная проект, плохо себе все представляешь, поэтому первые этапы очень важны. В конце концов, не станешь же строить за \$50 изгородь, чтобы стеречь лошадь, которая стоит \$10, поэтому я согласен – сначала надо разобраться, что ты хочешь сделать. Мне также нравится идея не бросаться нанимать людей, пока не уяснишь хорошенько, что они должны делать. Иначе они будут просто жевать табак за твои деньги, пока кто-нибудь не придумает, чем их занять.

Мне стало интересно, как Роско оценивает положение.

– Потом я добрался до графика работ и оценки сроков. Тут есть толковые ребята. Ну, например, Барри Боэм (Barry Boehm) и Уокер Ройс (Walker Royce) немного в этом разобрались. Хотя их кокосовая модель – это уже выпендрож.

Я поежился, но не стал поправлять его. В конце концов СОСОМО и «кос» звучат похоже. Кроме того, речь Роско постоянно приходится переводить на обычный язык, поэтому особых дополнительных усилий от меня не потребовалось.

«А потом я почитал неких Крухтена и Буча и даже наткнулся на твоё имя. Вот здесь», – сказал он, показывая мне страницу в замусоленной книжке Гради «Объектные решения».¹

Точно, он застал меня на месте преступления. Вот он я, черным по белому, на стр. 136.

Роско расправляется со всеми

– Вот тут написано, как ты и этот парень Крухтен заметили, что длительность итерации в неделях примерно равна квадратному корню из длины кода, который нужно написать, если измерять ее в тысячах строк.

Ну как тут спорить: это была точная цитата. Я подумал, что сейчас мне придется за это ответить.

– Я даже поговорил со своим приятелем архитектором, и он сказал, что есть особый термин для обозначения тысячи строк кода. Говорит, что это называется *KSLOC* – от «kilo source lines of code». Тьфу!

Роско явно разминался перед основным выступлением.

– Какая-то шайка бакалавров. Все равно что оценивать проект постройки пирамид и выбрать «кирпич» в качестве единицы измерения. Звучит нелепо, поэтому изобретается «килокирпич». Глупость, какой я еще не встречал.

Он явно набирал скорость, и я не хотел бы попасть под этот паровоз.

– Однако, как я выяснил в разговоре с этим архитектором, главная проблема в том, что никто не знает, как считать эти «слоки» или «килослоки». Считать ли все строки, включая комментарии. Считать ли заголовки вместе с файлами исходного кода реализации, в которых явно много повторов? И как быть с теми огромными библиотеками, которые вы просто берете, не создавая их с нуля? В принципе для получения этого числа можно воспользоваться услугой «молитва по телефону».

Я пометил себе, что нужно поговорить с архитектором и узнать, что он сообщил Роско о заголовках и прочем. Ясно, что тот не сам это выдумал.

¹ Booch, Grady. Object Solutions: Managing the Object-Oriented Project (Boston: Addison-Wesley, 1995).

Но у него была своя точка зрения. Мы состряпали формулу, основанную на числе, которое, если выразиться мягко, допускало разные интерпретации. Точность прогноза обусловлена нашим определением для SLOC – количества строк исходного кода.

Кое-что мы делаем правильно, часть вторая

Как только я решил, что Роско окончательно меня уничтожил, он вы- сказал мне в некотором роде комплимент.

– Кое-что ты уловил правильно, Джо. Все дело в квадратном корне. Идея в том, что итераций не должно быть слишком много, потому что начало и конец каждой из них связаны с накладными расходами, которые, как всем понятно, совершенно непроизводительны. С другой стороны, итераций не должно быть слишком мало, потому что тогда теряется весь смысл итеративной разработки. Как в сказке братьев Гримм про горшочек каши – все должно быть в меру. Я думаю, что ты был прав, связав длину итерации с квадратным корнем из размера проекта. И выбор недели в качестве единицы измерения тоже был правильным, поскольку это основная единица измерения труда. Ты ошибся только в том, что связался с этими «килословами». У меня есть гораздо более простой способ.

Я приготовился слушать. Со всем вниманием.

– Главная проблема в том, что ты неверно начинаешь. Надо взять эту косоугольную модель или воспользоваться каким-то другим приемом, чтобы договориться о том, сколько всего времени тебе дадут на выполнение проекта. Руководство всегда хочет получить все быстрее, и ты всегда будешь в итоге обсуждать конечную дату, а не число строк кода. Ты можешь в зависимости от даты выторговывать, скажем, численность команды или набор функций, но в конечном счете фиксированной останется только длительность проекта. Когда она определена, все становится просто. Длительность одной итерации должна быть равна квадратному корню из общей длительности проекта. Дай я тебе покажу свою маленькую табличку. – С этими словами он вытащил табл. 10.2.

Меня несколько не удивило появление календаря Роско. Его увлечение квадратными корнями не миновало и этих расчетов.

Черт возьми! Посмотрев в свою базу данных проектов, я обнаружил, что оценки Роско не очень с ней расходились. Как он любит говорить: – «Одной динамитной шашкой больше или меньше...».

И никаких «словес». Я был потрясен.

Таблица 10.2. Справочник Роско для вычисления длины итерации

Длительность	Длительность по Роско	Длина итерации	Количество итераций
Два года	100 недель	10 недель	10
Один год	49 недель	7 недель	7
Девять месяцев	36 недель	6 недель	6
Шесть месяцев	25 недель	5 недель	5
Четыре месяца	16 недель	4 недели	4
Два месяца	9 недель	3 недели	3
Один месяц	4 недели	2 недели	2

Роско допущен в компанию менеджеров программных проектов

По результатам этого маленького упражнения в определении длительности итерации я счел, что у Роско достаточно компетенции, чтобы приступить к первому итеративному проекту. Однако я сделал ему предостережение.

– Послушай, Роско, – сказал я, – ты столкнешься с тем, что заправлять программными проектами – это занятие невероятно щекотливое. Если у тебя возникнут какие-то идеи, я очень хочу, чтобы ты ими со мной поделился.

– Не сомневайся, – сказал он. – До встречи через месяц.

Резюме

Разумеется, с помощью старины Роско я пытаюсь добавить немного обычного здравого смысла в наше обсуждение управления разработкой программ. А для этого очень полезно по возможности избавиться от жаргона. Это может помочь и вам в разговорах с другими.

Я, конечно, еще не все сказал об оценках сроков. Практическая часть оценивания начинается с графика работ.

В следующей главе Роско высказывает свое мнение о графиках разработки ПО.

ГЛАВА ОДИННАДЦАТАЯ

График работ



Вот тут-то и начинаются все беды, потому что обычный менеджер программного проекта смотрит на свой график как на *рабочий документ*, а его начальник склонен считать этот документ *контрактом*. Это «тонкое» различие уже долгие годы оказывается причиной разных неприятностей.

В результате возникла позорная игра в «два графика», когда есть «внутренний график» с намеренно сжатыми сроками – чтобы разработчикам не казалось, что у них много времени, – и «внешний график», в котором для страховки сроки внутреннего графика увеличены, – для показа всем остальным. Честно говоря, мне всегда не очень нравилось такое мошенничество. Трудно держать в тайне наличие двух графиков, и когда она вскроется, доверие перестанут вызывать оба графика, как бы вы ни старались объяснять, что вот этот – для разработчиков, а вон тот – для бизнес-плана. Я рекомендую иметь один график и стараться сделать его максимально честным.

В итоге главный управляющий с трудом пытается договориться с менеджером программной разработки по поводу его графика, и точно так же менеджеру программного проекта бывает сложно разумно обсуждать с техническим руководителем какого-то участка график работы над создаваемой его группой компонентой или подсистемой. Причина в том, что говорить-то особо не о чем. Конечно, можно обсуждать, нельзя ли сделать что-то скорее, но это сводится к личному мнению, и оценки для каждого отдельного элемента редко бывают глубоко ошибочными. На самом деле все просчеты составления графиков происходят по двум главным причинам,

которые систематически проявляют себя в программных проектах. В первых, существуют взаимозависимости, о которых в начале проекта известно мало или вообще ничего. В результате оказывается, что некоторая часть проекта отстает, а вслед за ней тормозят и другие части, потому что их разработчикам оказалась нужна важная компонента, о чем они до того времени не подозревали. И эта зависимость обнаруживается только тогда, когда задерживается разработка этих вторичных компонент.

Хорошим противоядием для подобного отставания от графика является *итеративная разработка*. На ранних итерациях компоновка больших деталей и получение первого образца действующей системы сразу выявляет такие «скрытые» зависимости. Определив таковые, можно применить различные стратегии: принудительное устранение зависимостей, дополнительное внимание/ресурсы для критических компонент и т. д. Другой способ ослабить влияние этой проблемы состоит в том, чтобы предложить всем командам согласовать свои интерфейсы, прежде чем реализовывать внутренние механизмы. Благодаря этому другие группы смогут продолжать работу на основе уже опубликованных интерфейсов. Пока интерфейсы не меняются или меняются незначительно, внутреннее устройство можно существенно менять без особого ущерба.

Вторая, более коварная, причина срыва графиков состоит в том, что отставание происходит медленно и постепенно. Фредерик Брукс давным-давно отметил, что отставание проекта на год состоит из задержек на один день. Если первую контрольную точку вы пройдете с опозданием на неделю или две, то скорее всего вы уже *никогда* не наверстаете упущенное, и все последующие точки пройдете с таким же или еще большим опозданием. Это та пресловутая смерть от разрезания на тысячу кусков, и даже при самом усердном управлении ее трудно избежать. Но если вы, менеджер программного проекта, еще и ослабите внимание, то пеняйте на себя!

Роско формулирует задачу: насколько вы опоздаете?

Мы уже знакомы с Роско Леруа,¹ бывшим горным инженером, старым и сварливым. Был унылый дождливый день, когда он остановил свой пикап у моего дома и побежал к входной двери, хлопая на ветру дождевиком. «Вот и солнце всходит», – подумал я.

¹ См. главу 5 «Самое важное» и главу 10 «Оценка времени», где о нем кое-что рассказывается (если вы читаете книгу не по порядку и раньше не встречались с Роско).

Если вы помните, Роско – это старый боевой товарищ моего отца с богатым жизненным опытом, накопленным под тяжелыми ударами и жестокими превратностями судьбы. Я люблю послушать Роско, потому что у него свежий взгляд на вещи, и его ум не сдавлен шорами расхожих истин, общепринятых учений и теоретических изысканий. По моим сведениям, за свою «карьеру»¹ он помог не одному техническому менеджеру спасти свою шкуру.

– Ну, – начал я, – как продвигается новая карьера в управлении программными разработками?

«Так, похоже, что мы друг друга не понимаем»², – заговорил Роско. После такого начала у меня перед глазами возникли солнечные очки дорожного босса из «Хладнокровного Люка». Хотелось бы, чтобы конец был не таким, как в кино.

– Прежде всего, – сказал он, – программные проекты никогда не выполняются в срок. Я подчеркиваю: *никогда*. Это странно. В конце концов практически любые оценки всегда содержат ошибку в виде «плюс-минус столько-то». Но в программировании «минус» где-то затерялся. С таким же успехом автор оценки мог сказать: вот лучшее, на что мы способны.

Я согласился, что его наблюдение по большей части правильно. Роско устроил мне разнос за «по большей части». Он потребовал, чтобы я привел ему хоть один проект, законченный досрочно. Я поежился. Помню, как однажды мы прошли до срока одну контрольную точку, но чтобы весь проект...

«Ну что ж, мне кажется, что все проблемы с командованием сводятся к тому, чтобы выяснить, насколько же ты опоздаешь в действительности», – заявил он с улыбкой.

Джо отчасти возвращает свои позиции

Попав в довольно неловкое положение, я решил, что пора извлечь пользу из моего многолетнего опыта задержки программных проектов. Я докажу этому отставному землекопу, что он ни черта не знает.

¹ Если спросить у Роско, то он поднимет на смех понятие «карьеры». «Просто пытался доводить дело до конца, сынок», – вот что обычно отвечает Роско на вопрос о своих достижениях или неудачах.

² Известная цитата из фильма «Хладнокровный Люк» (Cool Hand Luke) с Полом Ньюменом в главной роли.

– На самом деле, Роско, – начал я спокойно, – не существует какой-то одной оценки. Есть исходная оценка в начале проекта. Затем по ходу дела вырабатывается следующая. И так оценки производятся постоянно вплоть до конца работы. Поэтому в действительности в каждый данный момент времени следует говорить о том, какая ожидается задержка *относительно последней выполненной оценки*. В конце концов, последняя оценка включает в себя все, что тебе стало известно к ее моменту.

«Я тебя понял, сынок, – отвечал Роско. – Но вот что я скажу. Эти оценки по ходу дела могли бы становиться более точными. По двум причинам. Во-первых, во время работы ты должен чему-то научиться и поумнеть, а, во-вторых, оставшихся дел становится все меньше и меньше. В начале у тебя куча неопределенностей и масса дел. Зато больше времени, чтобы успеть исправить ошибки. Тут мне надо еще немного подумать».

Я решил, что этот раунд закончился вничью. Роско допил свой кофе (черный без сахара) и несколько агрессивно затушил сигару. Уверен, что я дал ему и другую тему для размышлений. Мне было ясно, что он напряженно пытается понять, почему оценки сроков программных проектов оказывались настолько ненадежными в сравнении с другими проектами, которыми ему доводилось руководить.

Роско возвращается

Прошло немного времени, и Роско вернулся, чтобы продолжить наши разговоры. Похоже, что моим устоявшимся представлениям снова суждено было быть опрокинутыми.

На этот раз Роско был гораздо спокойнее.

– Как я сказал в прошлый раз, обычно мы опаздываем. Вопрос в том, насколько? Думаю, что я нашел решение.

– Как обычно, не обойтись без квадратных корней, а единицей измерения будет... правильно, неделя. Я полагаю, что при правильном управлении задержка проекта в неделях может составить корень квадратный из количества оставшихся согласно прогнозу недель. Поэтому, если ты говоришь мне, что осталось, например, 16 недель, я склонен считать, что на самом деле тебе понадобится от 16 до 20 недель.

Непостижимо. Опять квадратный корень? Возможно ли? У него что – никаких трюков не осталось в запасе?

– На самом деле тут есть пять разных случаев, – сообщил он далее. – Я разобрался со всеми.

На этот раз Роско приготовил к бою тяжелую артиллерию. Чем дольше он говорил, тем внимательнее я слушал.

– Прежде всего, возьмем проект с хорошим управлением, руководитель которого знает свое дело. В итоге он завершит работу согласно своей оценке или на корень квадратный из нее позже. Так вот, иногда этот парень может постараться и сделать действительно точную оценку, да еще и не спускать глаз с исполнителей. А там еще «высшие силы»¹ улыбнутся ему и т. д., и закончит он раньше. Может быть, даже на квадратный корень раньше. Бывает.

Я вспомнил, как был смущен, когда в последнюю нашу беседу не смог привести таких примеров.

– Это все хорошие случаи, – добавил он.

Список негодяев по Роско

Н о есть еще 75%. Они составляют три других случая.

– Прежде всего, кое-кто заканчивает работу быстрее чем за корень квадратный до срока. Этих хитрецов называют *перестраховщиками*, и они вредны, потому что приносят убытки – не в результате опоздания, а в результате завышенных оценок для всего. Единственный способ закончить раньше с превышением квадратного корня состоит в том, чтобы сжульничать в оценке. На самом деле уволить надо того босса, который согласился с этой оценкой. А если хорошенько подумать, то выгнать надо обоих.

У меня стало складываться впечатление, что мистер Леруа может совершить революцию в наших взглядах на эту проблему.

«Теперь возьмем тех, у кого задержка проекта составляет от одного до двух квадратных корней. Они не так безнадежны. Может быть, они просто неопытны, и им нужно время, чтобы отточить свои навыки оценивания и выполнения. Мы теряем на них деньги, но можно поработать с ними, чтобы они смогли укладываться в диапазон одного квадратного корня. Если у них есть немного сообразительности, можно натренировать их до требуемых результатов».

– Наконец, есть те, кто выходит с опозданием за пределы двух квадратных корней. Либо они не в состоянии управлять, либо не имеют представления о том, как делать оценки. При этом неважно, что именно они не умеют – оценивать или управлять. Единственное решение – расстаться с ни-

¹ Видимо, Роско имеет в виду вмешательство провидения.

ми, потому что катастрофы, возникающие по их милости, приведут тебя к разорению.

График Роско

После этого Роско гордо извлек диаграмму, показанную на рис. 11.1. Он был горд, что смог воспользоваться своей новой игрушкой – Microsoft Excel. «Хороших парней я пометил «очень хорошо» и «отлично». Задержки от одного до двух квадратных корней я пометил «нуждается в помощи». Два других патологических случая тоже ясно помечены».

«Вот оно. Граница между «очень хорошо» и «отлично» есть «вовремя». Все, что понадобилось, это один лист бумаги».

И Роско откинулся на стуле, как бы завершив излагать свои доводы.

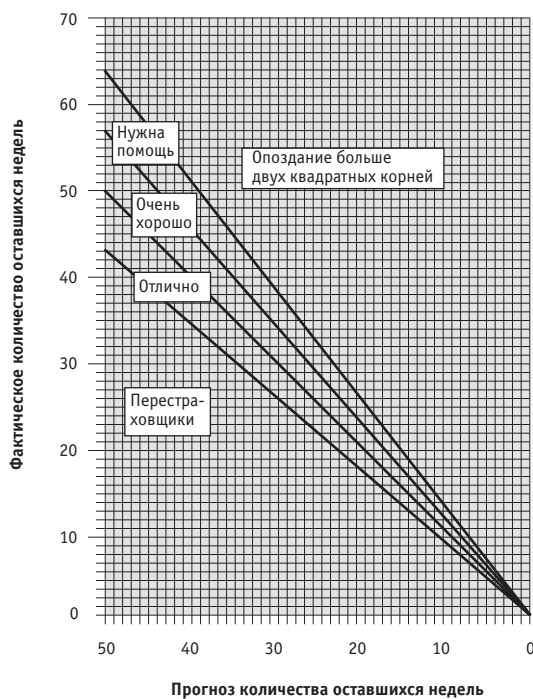


Рис. 11.1. Командование проектами

Последнее возражение

«Что ж, Роско, — отвечал я, — ты, как всегда, разобрался в сути. Мне понравился твой *количественный* подход к прогнозированию задержки. Но в твоей теории есть одно упущение».

Роско немедленно встрепенулся. Он любит, когда ему бросают вызов.

«Вот незадача: как я узнаю, к какой категории относится мой менеджер проекта? Если я знаю, что он ас в этом деле, то все в порядке. Но если я никогда не работал с ним раньше?»

«Ерунда, — ответил Роско, — это всего лишь задача начальной калибровки. Вот как я это делаю. Прежде всего, когда я хочу, чтобы кто-то что-то сделал, я прошу его оценить срок. И, *обрати внимание*, я записываю его».

Тут я сострил по поводу огрызка его карандаша, о чем тут же пожалел.

«Послушайте, мистер Умник, самый короткий карандаш будет подлиннее самой долгой памяти. И не забывайте, что когда Тайгер Вудс выходит на охоту за своими миллионами, он ведет им счет таким же коротеньким карандашиком».

Получив по заслугам, я решил закрыть рот и отверзнуть уши.

«Так вот, насчет программного обеспечения: я постараюсь привлечь их к нескольким подпроектам и заданиям разной длительности. Разумеется, в начале каждой итерации итеративного проекта необходимо оценить срок. И каждую оценку я записываю. А потом я смотрю, какого результата они добились».

«В результате я получаю кучу точек на своем графике прогнозов и фактических результатов. Кое-кто уверенно попадает в одну и ту же зону. Замечательно. Они прошли калибровку».

«Результаты других размазаны по всему полю. Я бы на твоём месте сел с ними и попытался разобраться. Либо через некоторое время они станут предсказуемыми, либо придется послать их подальше. И ты сам знаешь, что делать с категориями перестраховщиков и «больше двух квадратных корней». Только следи, чтобы дверь не хлопнула их по заднице, когда они будут выходить».

«Это средство весьма полезно для тех, кто попадает в зону «требуется помощь». Необходимо показать им, что они должны делать, чтобы мы стали зарабатывать деньги благодаря им, а не терять их. Это поставит перед ними цель, к которой надо стремиться. Они должны научиться не выходить за пределы одного квадратного корня, какой бы ни была их оценка срока».

«Видишь ли, Джо, – сказал он, сев поудобнее и улыбаясь, – мне не так важен результат, как его предсказуемость. Меня устроит задержка на один квадратный корень, лишь бы я знал, что это мои максимальные потери. Тогда я справлюсь».

Что бы я ни придумал, разделает меня под орех.

Заключительный выстрел Роско

«Кстати, – завершил Роско, – надеюсь, ты заметил, что происходит в моей модели, когда мы приближаемся к концу проекта». Я не заметил, но мне показалось, что это интересный предельный случай.

«Когда у тебя по прогнозу осталась одна неделя, то квадратный корень из 1 – тоже 1. Поэтому ты достиг предела точности метода. В пределах одной недели тебе всегда нужна еще одна неделя, потому что если ты честен сам с собой, то будешь находить дела с такой же скоростью, как вычеркивать их из списка. Поэтому даже теоретически проекты могут продолжаться до бесконечности, продлеваясь по одной неделе».

«Поэтому мы и говорим о «переходе к коротким ударам».¹ В последнюю неделю необходимо принимать решения особого типа и оценивать, сколько еще требуется времени, бессмысленно. Я тебя доведу до последней недели, но завершение проекта – это твоя работа».

Я очень надеюсь, что Роско не бросит свою новую карьеру. Он должен еще многому научить нас.

Резюме

В разговорах о графиках работ повышенный интерес у людей вызывали два момента. Первый – это понятие *предсказуемости*. Менеджеры неоднократно жаловались мне, что совершенно не выносят непредсказуемости. Роско говорит, что больше всего мешает, когда результаты очень сильно отличаются от плана. Если задержки носят систематический характер, то ситуация приемлема.

¹ Один из редких случаев, когда Роско упоминает гольф. Чтобы закончить лунку в гольфе, необходимо делать легкие удары по мячу. Обычно их называют «короткими ударами». Поэтому переход к коротким ударам указывает на завершающие действия для выполнения задания.

И это приводит нас ко второму вопросу – к идее *калибровки*. Чтобы обеспечить какую бы то ни было предсказуемость, необходимо проанализировать, едва ли не на уровне отдельного разработчика, архивные данные о качестве оценок. Выяснить, кто систематически проявляет излишний оптимизм, а кто систематически занижает оценки, очень полезно.

Мне кажется, что все хорошие менеджеры делают это на качественном уровне. Мысль Роско, приведенная здесь, сводится к тому, что мы должны собирать данные по ходу работы, *постоянно* осуществлять калибровку наших работников и уточнять ее. Уяснив, насколько можно доверять их оценкам сроков, мы могли бы создавать более надежные графики работы.

Любопытно, что эта глава оказалась очень компактной. Я объясняю это тем, что идеи Роско коротки и свежи. Он еще раз подтвердил, что хорошая идея не нуждается в многословии.

В следующей главе я рассматриваю любопытное явление. Во всех успешных проектах наблюдается некий внутренний ритм, которому они подчинены. Где его источник? Я предлагаю некую модель, которая может объяснить этот эффект.

ГЛАВА ДВЕНАДЦАТАЯ



Ритм

Одна из особенностей успешных проектов состоит в том, что они подчиняются некоему ритму. Точнее нескольким ритмам. Есть общий ритм перехода от одного этапа к другому в процессе итеративной разработки. Есть более короткий ритм, которому следуют итерации. Наконец, есть *ритм построения* – биение его пульса, прослушиваемое, например, раз в сутки. Эти ритмы – верный признак того, что «механизм проекта» работает. Конечно, иногда возникают сбои, но они представляют собой исключение на фоне ровного развития, о котором свидетельствуют устойчивые ритмы.

Напротив, менее удачные проекты как будто движутся резкими толчками. Все происходит произвольным и неравномерным образом. Долгое время кажется, что ничего вообще не происходит. Затем выпускается версия продукта, и наступает период оптимизма. В скором времени после этого снова обнаруживается уйма проблем и возникает упадок духа. Настроение людей колеблется вслед за часто меняющейся оценкой проекта и продукта. Когда проект отстает от плана, люди начинают работать интенсивнее, но ничего не меняется. Знакомая ситуация?

Я наблюдал такие противоположные сценарии развития много лет и размышлял, нет ли в их основе какого-то особого механизма, по крайней мере, для успешных проектов. Я решил, что путей, которыми неудачные проекты приходят к своему провалу, неисчислимо множество, а потому для них и механизмов таких бесконечно много. Зато отыскать единый механизм успеха, может быть, проще.

Взгляд физика на течение проекта

Случилось так, что я решил воспользоваться простой моделью, а именно вторым законом Ньютона. Поскольку вся классическая механика подчиняется этому простому принципу, не исключено, что существует «второй закон Ньютона» и для проектов разработки программного обеспечения. В конце концов, динамика кривой «процента выполнения» проекта обычно весьма единообразна во всех проектах. Не попытаться ли описать силы, действующие на проект, через производные этой кривой? Конечно, распространять законы физики на динамику проектов – дело довольно рискованное, но я рассчитывал, что помогу менеджерам проектов разобраться в ритмах своих команд.¹

А это вы уже видели?

Чаще всего график выполнения проекта, представляемый после его завершения, выглядит так, как на рис. 12.1.

Такая кривая выполнения для большинства проектов является *запланированной*. Обычно в отчете кривая выполнения выглядит как плановая, но реальность вносит в нее некоторые коррективы.

Если кривые выполнения имеют такой вид столь часто, то нет ли каких-то фундаментальных сил, действием которых объясняется эта форма?

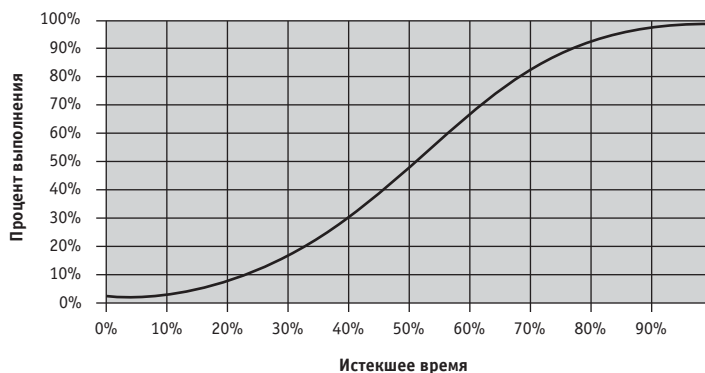


Рис. 12.1. Типичная кривая степени выполнения проекта

¹ Этот анализ в несколько сокращенном виде был опубликован в номере «Dr. Dobbs' Journal» (Журнал доктора Доббса) за август 2002 г.

Все началось с Ньютона

Часто утверждают, что «современная» физика началась с Ньютона, хотя первым современным физиком был, наверно, Галилей. Своим вторым законом Ньютон поведал нам, что ускорение тела пропорционально равнодействующей приложенных к нему сил, при этом коэффициентом пропорциональности является величина, обратная массе тела.¹ Чем больше сила, тем больше ускорение.

Беда в том, что ускорение мы испытываем редко. Мы ощущаем его, резко трогаясь с места на перекрестке, нажимая на тормоз, в начале движения лифта или при внезапной его остановке. Чаше же мы замечаем две другие переменные: *скорость* и *положение* в пространстве. Обычно мы знаем, что движемся быстро или медленно, так же как знаем, где мы находимся. Но сила связана с этими характеристиками лишь косвенным образом.

Производные, скорость изменения и наклон касательной

Если вы знакомы с дифференциальным исчислением, то знаете, что скорость – это первая производная положения, а ускорение – первая производная скорости, что делает ускорение второй производной положения.² Если же вы не изучали дифференциальное исчисление (или забыли его), то вместо «производной» поставьте «темп изменения». Таким образом, скорость – это темп изменения положения, а ускорение – темп изменения скорости.

¹ Под равнодействующей я понимаю сумму всех сил, действующих на массу. Чаше всего закон выражается формулой $\mathbf{F} = m\mathbf{a}$, где \mathbf{F} и \mathbf{a} выделены жирным шрифтом, поскольку это векторные величины. Для простоты я опускаю эту векторную нотацию, т. к. обычно рассматриваю одно измерение. Заметьте также, что в более общей форме второй закон записывается так: $\mathbf{F} = d\mathbf{p}/dt$, где \mathbf{p} – количество движения. Эта формула сводится к $\mathbf{F} = m\mathbf{a}$, если масса постоянна, что также предполагается мной на протяжении этой главы. Это очень важное допущение: расползание проекта соответствует увеличению массы проекта, которое сводит анализ на нет. С расползанием проекта надо бороться беспощадно, ведь даже мелкие расширения, накапливаясь исподтишка, могут погубить его.

² Имеется в виду производная по времени – понятие из курса средней школы. Могут быть и более сложные функциональные зависимости и, соответственно, их производные и для пространственных переменных, например, кривые годографа в фазовом пространстве в теории катастроф и их производные. – *Примеч. науч. ред.*



Рис. 12.2. Траектория шарика, брошенного вертикально

Согласно Ньютону, ускорение пропорционально равнодействующей приложенных сил, поэтому именно вторая производная положения объекта пропорциональна силе. Иными словами, чтобы получить данные о положении, надо дважды *проинтегрировать* приложенную равнодействующую силу. Большинству из нас это, конечно, *не совсем* ясно. Поэтому я пока оставлю в стороне все эти дела с двойным интегрированием и скажу, что для интегрирования и дифференцирования разработан математический аппарат, а наша задача – разобраться с механизмом.

Тем, кому удобнее иметь дело с графическим представлением, напомним, что найти производную или скорость изменения любой кривой можно по касательной к этой кривой в интересующей вас точке. Например, на рис. 12.2, где изображена парабола, производная, или скорость изменения, равна нулю в средней точке. Это видно из того, что в этой точке касательная горизонтальна, и потому ее наклон (или скорость изменения) равен нулю. До этой точки касательная всюду имела положительный наклон; после средней точки наклон становится отрицательным.

Простой физический пример

Для шарика, брошенного вертикально вверх, траектория полета определяется очень просто.¹ Если построить график высоты как функции

¹ В этом примере можно пренебречь сопротивлением воздуха. Как и во всех физических экспериментах, где делается такое допущение, последствия оказываются катастрофическими для экспериментатора: он гибнет из-за отсутствия воздуха для дыхания.

времени, получится все та же замечательная парабола, показанная на рис. 12.2.¹

Если теперь построить график скорости шарика,² то получится график, приведенный на рис. 12.3.

Обратите внимание, что я «нормализовал» вертикальную ось так,³ что скорость попадает в интервал между плюс и минус единицей, — это помогает понять суть, абстрагировавшись от чисел.⁴ График показывает, что вначале скорость движения шарика максимальна и равна одной «единице», а затем она линейно убывает, пока не достигнет нуля в средней точке

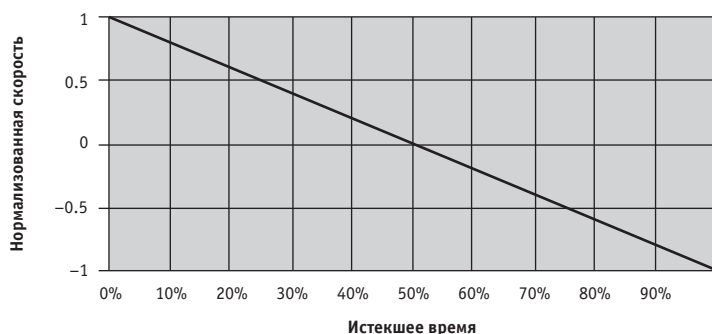


Рис. 12.3. Скорость шарика при бросании вверх

¹ Графики, приведенные в этой главе, я строил с помощью Excel. Для горизонтальной оси (времени в полете) выбрал интервал от 0% до 100%, а затем сгенерировал значения функции по формуле. Вертикальную ось я всегда нормализую к «единице», т. е. высота изменяется от нуля до максимума, который я называю «единицей». Все производные я получил численными методами, снова воспользовавшись Excel для расчетов и построения графиков. Благодаря этому вы можете повторить все самостоятельно, и для получения результатов не надо знать дифференцирование.

² Я делаю это путем численного дифференцирования точек параболической кривой. Это означает, что я беру последовательные разности между точками первой кривой и затем умножаю на константу, чтобы значения попадали в интервал между плюс и минус единицей. Внимательный читатель заметит, что при этом «пропадет» первая точка на графике первой производной и первые две точки на графике второй производной, а также возникнет маленький сдвиг (в полклеточки) графиков производных. Пока это совсем незаметно и никак не влияет на рассуждения, просто это сведения о том, как я вычисляю производные.

³ На стартовую скорость. — *Примеч. науч. ред.*

⁴ Здесь действует важное соглашение: направление «вверх» положительно. То есть положительный вектор скорости направлен вверх, а отрицательный вниз.

полета. Это понятно, потому что в середине полета достигается самая высокая точка, и в этот момент шарик не движется ни вверх, ни вниз, поэтому его скорость равна нулю. Затем он начинает падать, и его скорость становится отрицательной.¹ Когда шарик возвращается в исходную точку, его скорость точно такая же, как в момент броска, но движется он в противоположном направлении.

А что происходит с ускорением? Я просто повторяю этот процесс еще раз и получаю график, показанный на рис. 12.4.

Я снова «нормализовал» числовое значение, приведя его к -1 .² Мы видим, что ускорение постоянно. Это согласуется с нашей моделью: вблизи поверхности Земли сила гравитации постоянна. «Отрицательно» ускорение потому, что направление «вверх» выбрано положительным и сила притяжения тянет шарик «вниз».

Обратите внимание, что ускорение отрицательно на протяжении всего полета шарика, даже когда скорость положительна (полет вверх). В ходе полета мы выполняем «вычитание» из скорости. Мы начинаем с некоторого положительного значения скорости, которое уменьшается до нуля в са-

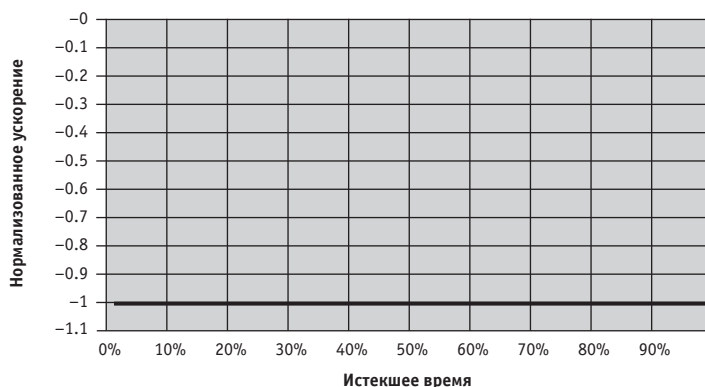


Рис. 12.4. Ускорение шарика при вертикальном бросании

¹ Необходимо различать скорость (velocity) и абсолютную скорость (speed). Абсолютная скорость всегда положительна; когда шарик падает, его абсолютная скорость растет. В то же время его скорость становится все более отрицательной. В обоих случаях абсолютная величина растет, но скорость (velocity) учитывает направление и потому становится отрицательной.

² Наш друг Excel настаивает на том, чтобы называть ноль «-0». Но мы-то знаем, что ноль только один и знака у него нет.

мой верхней точке, а затем продолжаем вычитание, пока скорость не станет в конце такой же, как в начале, но с отрицательным знаком.

Вот так мы подтвердили правоту Ньютона. Я «экспериментально» проверил, что график перемещения по параболе подразумевает постоянное ускорение, которое, в свою очередь, подразумевает постоянную силу (гравитации).

Можно ли с помощью подобных рассуждений понять, какие «силы» действуют на проект? Как и в случае ускорения, мы не чувствуем эти силы непосредственно. Вместо этого мы наблюдаем явления, связанные косвенно. Итак, вернемся к графику процента выполнения; он описывает происходящее, отображая результат совместного действия сил, которые мы не видим непосредственно. Что можно будет сказать о динамике проекта, если мы примем такой прототип графика степени завершенности проекта?

*График «степени завершенности проекта»:
положение в зависимости от времени*

Как я уже отмечал, большинство проектов имеет такой график, как на рис. 12.1.¹ Обратите внимание, что я снова выполнил нормализацию, чтобы по оси времени откладывались значения от 0% до 100% и по вертикальной оси также шли значения от 0% до 100%. Кроме того, я достаточно произвольно установил, что выполнение проекта на 50% соответствует 50% потраченного времени. Кривая сделана симметричной для простоты, что в данном контексте удобно. Пока у меня нет оснований делать иные допущения.²

График человеческого поведения

Вообще такие кривые называют *S-кривыми*, потому что если отступить назад и наклонить голову, они напоминают букву S. Они описывают мно-

¹ Сделаю ряд предостережений. Конечно, это идеализированный график. Реальные кривые не бывают такими гладкими. Кроме того, не все проекты движутся вперед, хотя график выполнения, в котором есть впадина, не часто увидишь *в отчете*. Но главная неприятность заключается в том, что нет какого-то одного показателя, отражающего «процент выполнения» от начала и до конца проекта. Отличить задокументированное положение дел от реального – одна из труднейших задач в любом проекте.

² Вам могут встретиться похожие кривые, но с другим «процентом выполнения» в середине временного промежутка. Это допустимо, но для простоты я их здесь не рассматриваю.

гочисленные случаи поведения людей; например, такой вид имеет классическая «кривая обучения». Кривая выполнения проекта напоминает кривую обучения, и это наблюдение должно навести нас на некоторые мысли.

Приступая к изучению чего-то нового, мы поначалу долго возимся, мало что понимая. Это соответствует первой, *пологой*, части кривой, когда время идет, а прогресс незначителен. Затем, когда мы разберемся, о чем идет речь, прогресс ускоряется, что находит отражение во второй части кривой, которую иногда называют *подъемом*. В этот период мы многое успеваем сделать за единицу времени, и если бы мы сумели сохранить такую скорость, то быстро решили бы задачу. Но мы неизбежно вступаем в третий этап, где кривая опять становится полой. Наступает *стабилизация*: больших успехов мы не достигаем, и последние 2% (изучение оставшихся противных мелочей) отнимают много времени. Такая картина обучения повторяется в разных областях знаний.

Если подъем плавный, то говорят о *легкой* кривой обучения; если подъем происходит очень быстро, то говорят о *крутой* кривой обучения. Многие не справляются с крутой кривой обучения: им требуется больше времени, чтобы усвоить большой объем новых знаний.

Интересны люди, *способные запоминать быстро*; у них очень короткий начальный период, затем очень крутой подъем, а потом они бросают учение, игнорируя оставшиеся несколько процентов. Укорачивая оба коротких участка и перемещаясь по крутому подъему, они могут сильно сократить время, необходимое им для приобретения новых знаний или умений. Но такие встречаются редко. Большинство из нас довольствуются двумя плоскими участками и умеренно крутым подъемом.

Проекты подчиняются аналогичному ритму. В начале проекта прогресс идет медленно. Много усилий тратится на планирование, организацию и исследования, что не дает заметного прогресса. Но постепенно вы выходите на подъем, который кажется довольно прямолинейным (хотя мы и знаем, что он не совсем прямой). В конце проекта снова происходит замедление, поэтому завершение всегда дается с трудом. Здесь также решение всех мелких проблем, связанных с окончанием проекта и выпуском продукта, отнимает относительно много времени. Поэтому проект, заверченный на 90%, формально таит не много риска, но до полного окончания может потребовать еще изрядное время. В конце работа замедляется, и сделать с этим практически ничего нельзя. Например, если подключить к проекту дополнительных людей, это только отдалит конечную дату, что было неоднократно продемонстрировано.

А как новый продукт принимается рынком? Та же картина. Сначала медленно, потому что люди должны узнать о продукте, выяснить его недостатки и преимущества, обсудить с коллегами и пройти процедуру отбора и приобретения. Это те, кто покупает первыми. Затем наступает подъем, когда продукт «наверстывает упущенное» и спрос растет до небес. Этот момент, когда продукт попадает в основной поток, Джеффри Мур (Geoffrey Moore) называет *прыжком через пропасть*.¹ Какое-то время это продолжается, а потом объем продаж снижается: мы вышли на третью, плоскую, часть кривой. Теперь продукт считается «зрелым», и покупают его в основном те, кто всегда покупает последним. Вполне вероятно, что это момент подъема для конкурирующего продукта, который отобрал часть рынка у более старого продукта. Вот так эта универсальная кривая представляет жизненный цикл торговли продуктом.

Кривая скорости проекта

Прежде чем вступить в бесконечный спор о том, как выглядит эта кривая «в действительности», просто примем, что она отражает большинство имеющихся данных о «проценте выполнения», и посмотрим, какие выводы можно сделать. Как и во многих других ситуациях в физике, сначала попробуем разобраться с идеализированным поведением, а потом откорректируем наши выводы, учтя в модели сопротивление воздуха и прочие природные факторы.

Важно отметить, что эта кривая представляет собой график зависимости координат от времени. Иными словами, это график нашего положения в каждый момент времени. А как будет выглядеть кривая «скорости»?

Скопировав наши предыдущие шаги, мы сможем ответить на этот вопрос. Вычислив скорость изменения графика процента выполнения проекта, получим кривую, представленную на рис. 12.5.

Вы должны были уже привыкнуть к тому, что данные по оси ординат нормализуются. Этот график показывает, что проекты начинаются медленно и потом некоторое время движутся быстро. Затем они достигают максимальной скорости (на рис. 12.5 максимальная скорость достигается

¹ Moore, Geoffrey A., и Regis McKenna «Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers, Revised Edition» New York: Harper Business, 1999.

Джеффри Мур, Маккенна, Р. «Преодоление пропасти. Маркетинг и продажа хайтек-товаров массовому потребителю». – Пер. с англ. – Вильямс, 2006.

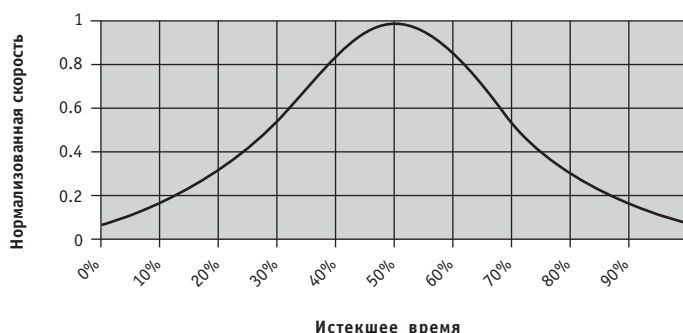


Рис. 12.5. Скорость кривой процента выполнения проекта

в середине пути, но только потому, что наша кривая выполнения проекта симметрична), а затем начинает снижаться. Пересекая финишную линию, вы движетесь довольно медленно, примерно так же, как в самом начале.

Это согласуется с нашим интуитивным повседневным опытом. Проекты медленно разгоняются, в какой-то момент «набирают скорость», а потом «вязнут в трясине» и движутся медленнее. Завершение сопровождается трудностями. Устранение мелких неувязок, необходимое для выпуска продукта, кажется бесконечным. Иногда возникает такое ощущение, будто пересекаешь финишную линию, едва держась на ногах.

Повторю, что точные значения для этой кривой могут быть различными, но в целом схема удивительно повторяется из одного проекта в другой. И какой из этого можно сделать вывод о скрытых «силах», управляющих проектом?

Физик в такой момент взял бы еще одну производную. Так же поступим и мы.

График сил, действующих в проекте

Получаемая в результате кривая ускорения, которая выражает действующие силы, показана на рис. 12.6.¹

Какие выводы можно из этого сделать?

¹ Здесь замечен небольшой сдвиг, о котором говорилось в примечании на стр. 180, — график сил должен пересекать ось в точке 50%. Напомню, что это следствие грубости того способа, которым я вычисляю производную, которое не имеет значения для главной темы.

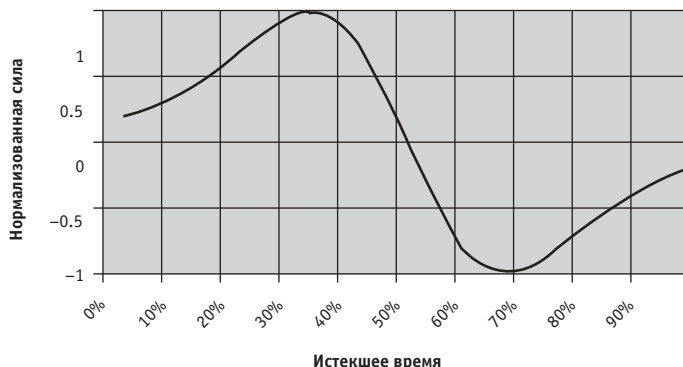


Рис. 12.6. Равнодействующая сил, демонстрируемая ускорением кривой процента завершения проекта

Вот одна из возможных интерпретаций. В первой трети проекта наблюдается возрастание положительной силы. Это соответствует большому воодушевлению в отношении нового проекта, появлению в команде новых участников и общему оптимизму относительно шансов на успех. Можно цинично охарактеризовать этот период словами «меньше знаешь — лучше спишь». Действует положительная возрастающая сила, благодаря которой проект набирает скорость (как иногда говорят, *кинетическую* энергию).¹ Примерно через одну треть пути эта положительная сила начинает убывать. Обстоятельства диктуют: прошло достаточно времени, чтобы люди поняли, каковы реальные трудности, и начали ощущать давление графика работы. Ведь прошла уже треть отпущенного срока, а работы осталось невпроворот. Кроме того, начинают ощущаться тяготы работы большой группой: много времени уходит на совещания, а доводить информацию до всех членов команды становится труднее.

Затем мы достигаем середины пути. В этом месте согласно показанному графику начинает действовать отрицательная сила. Команда ощущает, как что-то тянет проект назад. Многие действительно трудные проблемы поддаются решению не так быстро, как мы рассчитывали. По мере того как часы ведут неумолимый отсчет времени, все чаще людей охватывает страх. Когда пройдено примерно две трети пути, эта отрицательная сила достигает своего наивысшего значения; возникает ощущение, будто плывешь в вязкой жидкости. Если проект задерживается здесь слишком долго, он погибает.

¹ Похоже, что физика вмешивается в наш язык вопреки моим желаниям.

И затем, когда мы больше всего нуждаемся в этом, в проекте происходит перелом. Возникает прорыв, и внезапно все перестает выглядеть таким мрачным.¹ Хотя отрицательная сила все еще действует (все понимают, что предстоит доделать уйму вещей, а времени для этого осталось мало), она ослабевает. Главная причина в том, что команда увидела впереди финиш. Отрицательное действие продолжает убывать до самого пересечения финишной линии.

Учтите, что действительные «поворотные точки» на этой кривой – одна треть, половина и две трети – различны в разных проектах и оказывают влияние на кривую скорости и кривую процента выполнения проекта. В этих точках нет ничего магического, они обусловлены симметрией кривой степени завершенности проекта, которую я изначально выбрал для простоты изложения. Точное их расположение в нашем проекте неизвестно: можно лишь «предсказать», что проект пройдет эти фазы.

Это немного обнадеживает. Я начинаю с прототипа кривой процента завершения, беру пару производных и делаю вывод о силах, управляющих проектом. Эмпирические данные оказываются в согласии с теорией. Удалось ли мне тут чего-нибудь достичь?

Вмешательство реальности

Физик рассматривает теорию с нескольких разных точек зрения. Опасно экстраполировать законы, действующие на такие физические категории, как координаты, скорость и ускорение, на группу динамических переменных типа «процент завершения проекта», «скорость проекта» и «силы, действующие на проект». Необходимо тщательно следить за тем, чтобы не приписать модели больше «науки», чем в ней есть в действительности. Однако похоже, что аналогия дает результаты, согласующиеся с реальностью. Итак, перейдем к следующему шагу.

Физик судит о теории по ее способности объяснять и предсказывать. Таким образом, сначала теория должна объяснить результаты всех известных экспериментов. Если экспериментальные результаты противоречат теории, значит, теория неверна, если только не удастся показать, что экспериментатор сделал ошибку. В нашем примере для большинства проектов наблюдалась схожесть кривых процента выполнения, и проекты отличались лишь положением переходных точек на графиках производных.

¹ Какой-нибудь резкий менеджер в этом месте вполне может пробормотать что-нибудь вроде: «Ну вот, кажется, и все».

В результате мы приходим к ограниченной полезности теории. Но хотелось бы иметь возможность *предсказывать* поведение кривой процента выполнения, находясь на некоторой стадии незавершенности. Или, иными словами, мы хотим знать, когда можно ожидать завершения. В каждый данный момент мы располагаем лишь той частью кривой, которая находится «позади», и некоторым представлением о графике скорости. График «сил» довольно трудно определить количественно. На самом деле часто трудно истолковать даже график скорости, хотя метрики для скорости были бы весьма и весьма полезны. Знать свое «местоположение» важно, но для более точного прогноза необходимы как фактическое положение, так и фактическая скорость.

В настоящее время многие наши метрики проекта сосредоточены исключительно на местоположении: «Где мы находимся?» Как видно из данного изложения, наличие метрик скорости (изменения положения за единицу времени) имеет такое же большое значение. А если нам удастся также разобраться с тем, как изменяется скорость, то у нас возникнет еще более полная картина. Примерно так и следует рассматривать совокупность метрик проекта.

А что с итеративной разработкой?

Все вышесказанное плавно подводит нас к мысли о том, что проекты выполняются не одним махом, а разбиваются на итерации. У каждой итерации есть собственный ритм, поэтому описанные в данной главе графики представляют собой не более чем приближения. Так, для проекта с четырьмя итерациями график выполнения будет скорее похож на тот, который представлен на рис. 12.7.

Здесь мы наблюдаем четыре S-кривые, наложенные одна на другую. Степень выполнения кумулятивна, и я предположил, что каждая итерация занимает 25% времени и перемещает нас на 25% вперед вдоль графика степени завершения проекта. В последующем обсуждении я откажусь от этих упрощающих предположений.

Теперь посмотрим на графики скорости и ускорения (сил), соответствующие этому графику степени завершения в четырех итерациях (рис. 12.8).

О чем говорят нам эти графики производных?

Мы видим, что кривая скорости повторяет саму себя четыре раза, и это не удивительно. Проект набирает и теряет скорость четыре раза соответственно тому, что во всех итерациях ритм примерно совпадает. График

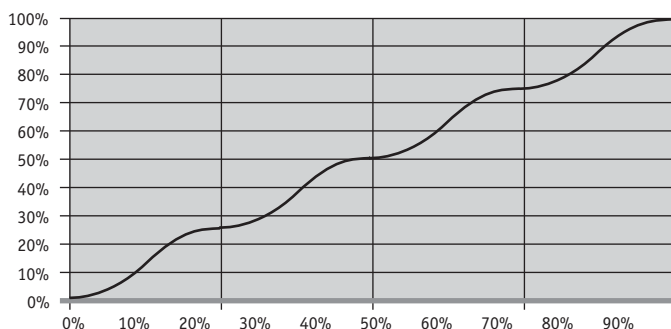


Рис. 12.7. График процента выполнения для проекта с четырьмя итерациями

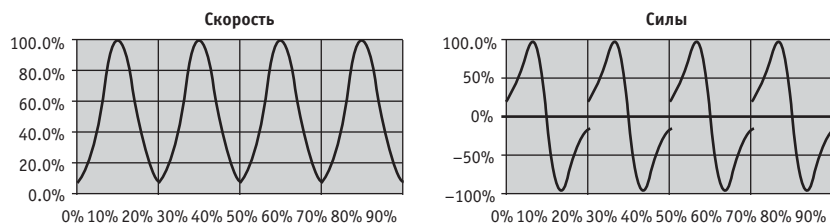


Рис. 12.8. Скорость и силы в проекте с четырьмя итерациями

второй производной – ускорения¹ – имеет три разрыва, которые также можно «видеть» на кривой скорости.² Это означает, что в начале каждой итерации, кроме первой, необходимо преодолеть действие отрицательной силы, сохраняющееся от предыдущей итерации, и приложить начальную положительную силу. В результате вектор скорости резко меняется. Без этой мгновенной прерывистой силы нельзя начать набирать ход, не потеряв при этом времени.

Большинству хороших менеджеров это известно, и в нужный момент они применяют необходимую силу. Не стоит слишком удивляться наличию этого разрыва – в конце концов, мы запустили проект в работу в нулевой

¹ К этому времени вы должны были уже привыкнуть к взаимозаменяемости обозначений «ускорение» и «сила».

² Если приглядеться, то обнаружится различие между четырьмя пиками со значением 100% на кривой скорости и тремя минимумами со значением 10%. Переход через максимум плавный, и производная в нем равна нулю. Напротив, переход через минимум не гладкий, и угол наклона резко меняется. Отсюда и разрыв на графике ускорения (силы).

точке с помощью ненулевой положительной силы. Я лишь хочу сказать, что в конце итерации у нас сохраняется остаточная отрицательная сила и необходимо скачком перейти к той положительной силе, с которой мы начали.

Применив эту положительную силу, надо опять наращивать ее, пока итерация не «упрется в стену» и не вступят в действие неизбежные отрицательные силы. После чего остается лишь осуществить прорыв для этой итерации, с помощью которого график отрицательной силы снова изменится на положительный.

Итак, мы знаем, как применить нашу модель к проекту с несколькими итерациями. Все остается по-прежнему: строим график степени завершения, берем производные и интерпретируем результаты. Разница лишь в том, что в начале кривая завершения немного сложнее. Но правила игры прежние.

Но когда все итерации выглядят одинаково, это скучно, не говоря уже о том, что у нас нет никакого представления о реальном проекте, с которым мы работаем, поскольку до сих пор наш анализ никак не отражал действительность. К ней мы теперь и обратимся.

Итерации и фазы

Реальная разработка может состоять из многих итераций, составляющих отдельные фазы проекта. Как отразить в модели такую особенность?

Я не собираюсь строить график, составленный из нескольких фаз и нескольких итераций в каждой фазе. Может быть, такая сложность вообще не нужна. В конечном счете наибольшие различия существуют между фазами, а не между итерациями внутри них. Поэтому, чтобы избежать чрезмерной сложности, создадим модели фаз и предположим, что в каждой фазе есть лишь одна итерация.

В методологии *Rational Unified Process* определены четыре фазы:¹

- Обследование (Inception)
- Проработка проекта (Elaboration)
- Построение системы (Construction)
- Передача в эксплуатацию (Transition)

Характеристики этих четырех фаз различны. В начале проекта проводится обширное исследование (обучение). В середине проекта мы больше

¹ Kruchten, Philippe, «The Rational Unified Process: An Introduction, Second Edition», Boston: Addison-Wesley, 2000.

занимаемся изобретением, чем исследованиями, хотя обучение по-прежнему интенсивно. На поздних стадиях проекта мы занимаемся тем, что требуется для «завершения»: отчасти изобретением, но в основном реализацией. По мере приближения проекта к концу обучение несколько ослабевает.¹

Пора разделить понятия «обучения» и «завершения». До сих пор я предполагал, что это одно и то же. В табл. 12.1 приведены цифры, которые, по мнению моего коллеги Филиппа Крухтена (Philippe Kruchten)² характерны для каждой из четырех фаз.

Таблица 12.1. Метрики обучения и завершения в методологии Rational Unified Process

Фаза	Истекшее время, %	Обученность, %	Завершенность, %
Обследование	0–10	0–10	0–5
Проработка проекта	10–40	10–60	5–25
Построение системы	40–90	60–90	25–90
Сдача в эксплуатацию	90–100	90–100	90–100

Из этих чисел следует, что при проведении разработки по фазам и итерациям мы обучаемся быстрее, чем завершается проект. Благодаря такому ускоренному обучению снижается риск. Как ввести эти соображения в нашу модель?

Достаточно всего двух изменений:

1. Начертить отдельные графики «процента обучения» и «процента завершения».
2. Для каждого из этих графиков начертить S-образные участки согласно табл. 12.1, что смоделирует реальность с данным уровнем приближения.

Результаты

Все результаты приведены на рис. 12.9.

¹ Здесь стоит упомянуть книгу Гради Буча «Object Solutions: Managing the Object-Oriented Project», Menlo Park: Addison-Wesley, 1996, p. 61.

² Данные по «завершению» получили Филипп Крухтен и Уокер Ройс в 1999 г., работая над Rational Unified Process. См. примечание на стр. 189. Данные по «обучению» взяты из частного разговора с Филиппом и представляют собой его экспертную оценку.

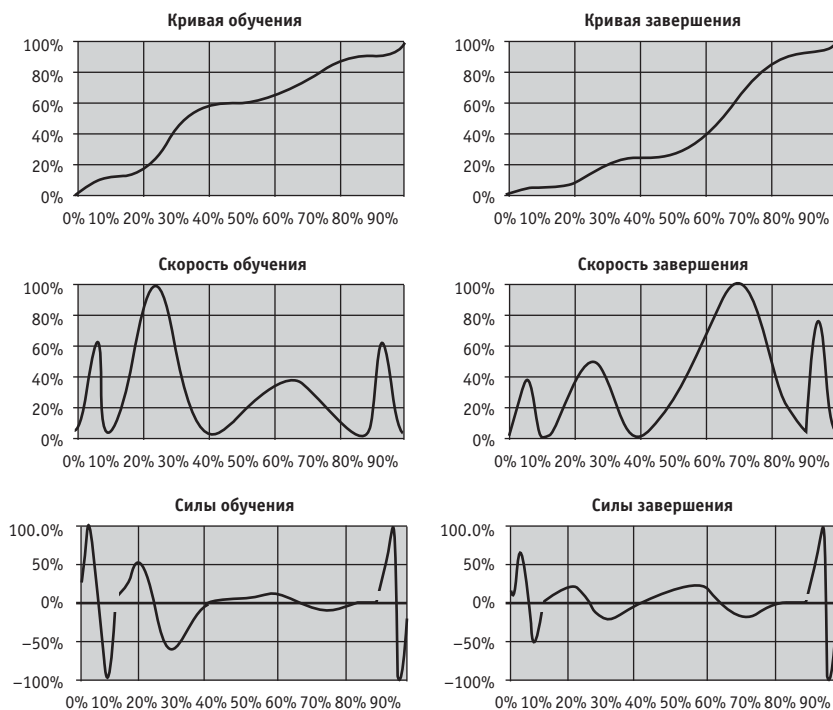


Рис. 12.9. Графики для табл. 12.1 – кривые обучения и завершения

Обсуждение результатов

Графики дают обширный материал для обсуждения.

Сначала займемся различием в форме кривых обучения и завершения. В принципе 60% необходимых знаний вы получаете в течение первых 40% срока, запланированного для проекта, и к этому моменту проект оказывается завершен всего на 25%. Это и понятно: при итеративной разработке обучению отводится более значительная роль, что преследует цель снизить риск. У этой медали есть обратная сторона, которая состоит в том, что не видно значительного «ощутимого» прогресса, поскольку обучение часто не сопровождается созданием заметного количества материальных объектов. Во всяком случае по истечении 40% отпущенного срока менеджеру проекта предстоит примерно такой разговор:

Босс: Вы потратили 40% времени, а проект завершен всего на 25%. Дело плохо.

Вы: Не совсем так. При итеративной разработке в первой половине проекта важно накопить знания. И мы выяснили 60% того, что собирались.

Босс: Вот как? Замечательно. Не покажете ли вы мне эти 60%?

На этот случай следует иметь дополнительные аргументы, которыми можно прикрыться. Я обычно демонстрирую список рисков, составленный в самом начале, и показываю, что обучение устранило или уменьшило эти риски.

Теперь посмотрим на кривые скорости. Отложим пока обсуждение первой и четвертой фаз и связанные с ними силы, поскольку интереснее обсуждать их одновременно. В данный момент сосредоточимся на фазах проработки и построения системы, между которыми есть важные различия.

Мы видим, что скорость обучения достигает максимума на этапе проработки, тогда как скорость завершения достигает максимума на этапе построения. Это согласуется с нашей моделью, в которой на этапе проработки мы отдали предпочтение накоплению знаний, в какой-то мере пожертвовав завершением. Идея в том, что реальные свидетельства завершенности проекта начинают появляться только на фазе построения системы, поэтому лишь в ней соответствующая кривая достигает своего пика.

Может возникнуть вопрос, почему обе кривые скорости опускаются почти до нуля на отметке 40%, а потом поднимаются снова. Не было бы лучше «сохранить» какую-то скорость при пересечении этой границы? Да, конечно, но мы не можем этого сделать, потому что на границе соединяются две S-кривые. По этой же самой причине на границах появляются разрывы графиков сил. В реальности этому математическому факту соответствует возникновение в любом проекте срыва при переходе между фазами. Из-за этого многие опытные менеджеры стараются сгладить прохождение границы и иногда создают для этого «передовой отряд», который начинает работать над следующей фазой с некоторым упреждением (до фактического завершения предшествующей фазы). Осуществить это трудно. Другим логическим следствием является создание дополнительных границ при увеличении количества фаз и итераций внутри каждой из них. Если переход каждой границы связан с постоянными накладными расходами, то при увеличении количества фаз и итераций суммарные издержки растут.

Сравнение сил, действующих в фазах проработки и построения системы, также довольно очевидно: силы обучения более значительны во время проработки и относительно слабее во время построения. Отмечу, что, на-

против, силы завершения достигают примерно одинаковых максимальных значений на фазах проработки и построения, что отраднo.

Теперь рассмотрим фазы обследования и передачи в эксплуатацию. Каждая из них занимает небольшой промежуток времени – всего лишь по 10% длительности всего проекта. Поскольку S-образность кривой приводит к наличию пика в ее производной, сжатость ее по времени приводит к большей высоте пика. Соответственно, кривая силы, необходимой для изменения кривой скорости в укороченный промежуток времени, также требует больших пиковых значений. Поэтому, обращаясь к графику сил для обучения и завершения, мы обнаруживаем в этих интервалах максимальные пики сил. Это следствие сжатости интервалов времени. Этому можно было бы противодействовать, изменяя поведение кривой так, чтобы она в эти короткие промежутки времени не была S-образной. Если бы графики в этих интервалах были просто линейными, то потребовались бы меньшие силы. S-кривые и их производные более пригодны для протяженных интервалов времени. Вероятно, они представляют собой хорошую модель для длительных интервалов и плохую для коротких.

С другой стороны, эти «значительные» силы могут быть отражением реальности, а не просто явлениями, возникающими при сжатии S-кривых во времени. Нет сомнений, что в конце проекта, во время передачи в эксплуатацию, на всех его участников действуют значительные силы. Это подтвердит каждый, кому приходилось завершать проект: окончание проекта дается так тяжело, потому что нужен сильный рывок, когда у команды уже остается совсем мало сил. Значительные силы также действуют в начале, на фазе исследования, потому что по окончании этой фазы приходится в первый раз принимать решение «идти/не идти» дальше. Если в этой точке участники проекта подумывают о том, чтобы все бросить, то можно не сомневаться, что придется задействовать значительные силы.

Последний график

Есть еще один график, который может представлять интерес. Допустим, что мы можем «сложить» вместе силы завершения и обучения. Силы представляют собой векторы, и я делаю несколько произвольное предположение, что силы завершения и силы обучения коллинеарны, т. е. действуют вдоль одного направления. Однако при таком предположении я могу арифметически сложить силы и получить суммарную действующую на проект силу как функцию истекшего времени, что показано на рис. 12.10.

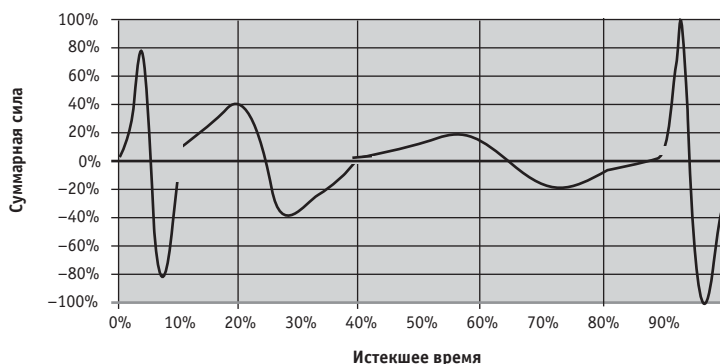


Рис. 12.10. Суммарное действие сил на проект (обучение + завершение)

Рассмотрим некоторые площади, ограниченные кривыми в четырех областях.¹ Отметим, что все участки имеют одинаковую форму, поэтому возьмем пиковое значение суммарной силы и продолжительность участка; их простое произведение будет пропорционально площади (табл. 12.2).

Таблица 12.2. Произведение пикового значения суммарной силы на длину интервала

Фаза	Процент истекшего время	Длина интервала времени	Пиковое значение суммарной силы	Произведение (импульс)
Обследование	0–10	10	0,8	8
Проработка проекта	10–40	30	0,4	12
Построение системы	40–90	50	0,2	10
Сача в эксплуатацию	90–100	10	1,0	10

Тем, кто любит математику и физику, напомним, что площадь является интегралом, а интеграл силы по времени равен изменению количества движения.² Эти данные показывают, что изменение количества движения примерно одинаково в каждой из четырех фаз. Изменение количества

¹ Я считаю здесь все площади положительными. Это означает, что я беру абсолютное значение площади под осью и не рассматриваю «отрицательные» площади. Если бы я этого не сделал, суммарная площадь оказалась бы нулевой.

² Более точно, интеграл силы по времени является импульсом, который равен изменению количества движения.

движения (или импульса) одинаково при построении системы и сдаче в эксплуатацию и равно 10 единицам. Наибольший импульс (во время обследования) равен 12 единицам. А суммарный импульс первой и второй фаз – 20 единиц – равен суммарному импульсу третьей и четвертой фаз. Половина суммарного импульса прикладывается во время первых 40% всего времени. Это также выражает идею «фронтальной загрузки», которую мы считаем полезной.

Обратите внимание, что разрыв кривой суммарной силы на рубеже 40% невелик. Это тоже может служить хорошим показателем. Поэтому выводы, которые можно сделать из данных Крухтена и Ройса по процентам обучения и завершения в четырех фазах проекта, согласуются с преимуществами, которые мы усматриваем в итеративных проектах разработки.

Я сделал массу допущений и подправил модель, чтобы мне было удобнее с ней работать, но полученные результаты согласуются как между собой, так и с реальными наблюдениями.

Резюме

Я взял простую универсальную модель S-кривой и применил ее к управлению проектами разработки ПО. С помощью обычной классической физики я перешел от S-кривой к скорости и ускорению, а, следовательно, и к силам, действующим в проекте. Я развил модель, охватив ею многофазовую итеративную разработку, и пришел к выводу, что результаты согласуются с наблюдениями. Я даже попытался разделить метрики обучения и завершения и проанализировать эти силы порознь. Наконец, я сложил вместе силы и пришел к некоторым выводам относительно суммарной силы как функции времени и импульса, прилагаемого к проекту на разных фазах.

Этот анализ в значительной мере умозрителен, поэтому не стесняйтесь и предлагайте собственные интерпретации нарисованных мной графиков. Надеюсь, что кто-то из вас найдет здесь зерна истины, не замеченные мной или моими рецензентами.

Между тем эта теоретическая модель помогает уверенно справляться с разными фазами проекта. Мысль о том, что в проекте наличествует некоторый «детерминизм», может подействовать успокаивающе, когда проект застопоривается и вас беспокоит возможное падение духа в команде.

Эта глава завершает третью часть книги. Теперь я перехожу к части IV, в которой изучаю человеческий фактор в управлении программными разработками.

ЧАСТЬ ЧЕТВЕРТАЯ

Человеческий фактор

В четвертой части книги фокус переносится с управления проектами на влияние человеческого фактора. Здесь я обращаю внимание на то, что ПО создают люди, и поэтому важно понимать, что мешает мотивировать их, а что помогает.

Глава 13 «Политика» посвящена жгучей теме. Инженеры, как оказалось, в целом неприязненно относятся к вторжению этого нетехнического элемента в их личное рабочее пространство. Я пришел к выводу, что особой чувствительностью в этой сфере обладают инженеры-разработчики ПО.

В главе 14 «Переговоры» я непосредственно обращаюсь к теме общения с разработчиками ПО. Существует неявный «протокол», помогающий избежать трений и разного рода недоразумений.

С тех пор, как Трейси Киддер (Tracy Kidder) ввел в своей книге «The Soul of a New Machine» (Душа новой машины) понятие «signing-up» (получение согласия), оно стало одним из главных элементов нашей отрасли. В главе 15 «Получение согласия» я попытался изучить подлинный смысл принятия на себя обязательств.

Наконец, в главе 16 «Вознаграждение» я затрагиваю щекотливую тему оплаты труда. Большинство моих знакомых, связанных с разработкой ПО, зарабатывают себе этим на жизнь, поэтому тема вознаграждения для них весьма актуальна. Многие менеджеры программных проектов чувствуют, как их сковывают политические интриги и вопросы оплаты, навязанные

им отделами по управлению персоналом. Затронуты и другие общие и оригинальные идеи, касающиеся этого предмета.

В этих главах я снова делаю попытку показать, что отличает друг от друга тех, кто связан с разработкой ПО, и способы эффективного взаимодействия с ними, учитывающие эти отличия.

ГЛАВА ТРИНАДЦАТАЯ

Политика

В главе 8 «За дверь его!» я говорил о налаживании процесса периодической сборки. Некоторые менеджеры программных проектов, читавшие ранние версии этой главы, пеняли мне, что я недостаточно осудил «политиканство» как один из главных неблагоприятных факторов. Из этого я сделал вывод о том, что роль политики неправильно понимают не только в разработке ПО, но и в управлении техническими организациями в целом.

Почему политика представляет собой такую чувствительную субстанцию? Раз уж она существует, то можно ли с ней как-то разобраться?

Большинство встречавшихся мне менеджеров программных проектов придерживалось в данном вопросе прямо противоположных подходов. Одна часть считает, что политика – это отравя, и стремится выкорчевать в своей организации все, от чего хоть немного пахнет «политикой». Такой квазирелигиозный пыл – явление по-своему нездоровое. Другая часть делает вид, что никакой политики не существует вовсе. Конечно, от того, что ее игнорируют, она никуда не исчезает, и они пытаются справиться с тем, что они же стараются представить фикцией. Неудача ждет и тех, и других.

Конечно, всякий разумный подход к этой проблеме предполагает, что сначала мы примем общее определение самого этого слова. Без общего рабочего словаря мы далеко не уедем. Поэтому прежде чем заняться этой противоречивой темой, я определю некоторый контекст.

Контекст

Когда мы начинаем разговор о таких аспектах человеческого поведения, как «политика» или политический процесс, то вступаем на минное поле культурных различий. Короче говоря, приемлемость или неприемлемость того или иного поведения оказывается разной в различных культурных контекстах. Поведение, которое в одной культуре проходит по грани дозволенного, в другой культуре может считаться настолько возмутительным, что там вы его никогда и не встретите. Когда далее в этой главе пойдет речь о «хорошей политике» и «плохой политике», будет иметься в виду норма, принятая в Северной Америке, и даже в этом случае она может иметь локальные оттенки. Некоторые из описываемых типов поведения могут оказаться более или менее приемлемыми и в европейском или азиатском контексте. Я буду также говорить об особом отношении к политическому процессу технических специалистов (и организаций, в которых они работают), которое, по моему мнению, выходит за рамки национальных или культурных границ. Я подметил некоторые общие тенденции, не зависящие от географии и культуры.

Высоко квалифицированные технические специалисты нередко испытывают отвращение к политике и политическим деятелям. Я не психолог, но причина, по-моему, в том, что они считают технические проблемы точными (черно-белыми), тогда как политика – это сфера неупорядоченная (характеризуемая оттенками серого). Благодаря своему образованию и подготовке ученые и инженеры в своей работе придерживаются акцентированного аналитического подхода, нацеленного на решение задачи и основанного на «базовых принципах» и применении данных в ходе различных индуктивных и дедуктивных процедур. Часто это позволяет находить изящные и законченные решения. А поскольку многие технические менеджеры вышли из рядов ученых и инженеров, они прихватили с собой не только их мощный инструментарий, но и свойственную им предвзятость.

В политике очень значителен элемент неопределенности, что часто вызывает у технических специалистов ощущение тревоги. Это вынуждает их как бы «играть на чужом поле», где они чувствуют себя в невыгодном положении, поскольку их технические знания приносят мало пользы. Напротив, опытные менеджеры привыкли не смущаться неопределенностью и достаточно легко ее переносят.

Возможно, это чрезмерное упрощение, но можно достаточно уверенно утверждать, что технические специалисты предпочитают решения, продиктованные только технической целесообразностью или хотя бы каким-то сочетанием технических и деловых целей. Чему они противятся, подчас яростно, – так это махинациям во время различных дискуссий и переговоров, связывая их с низостью и недостойностью человеческого поведения и считая, что при этом часто преследуются личные и карьерные соображения. Технические специалисты готовы заклеить любые такие действия как «политику» и, обобщив их подобным образом, придать им очень негативный оттенок.

Однако политика, какова бы ни была ее истинная сущность, не исчезнет лишь оттого, что она вам ненавистна. Наши желания и реальность – это совсем не одно и то же. По иронии судьбы некоторые рассматривают преодоление разрыва между тем и другим уже как некий политический процесс.

Я считаю, что тот, кто работает с людьми, неизбежно столкнется с политикой. Все люди думают по-разному, и политический процесс необходим для примирения различных мнений. Поэтому не будем осуждать политику, а попробуем разобраться в этом эффективном средстве разрешения неизбежных разногласий.

Определение

О тметим, что слово «политика» обозначает один из видов человеческой деятельности, таких же как «плотницкое дело», «игра на сцене» или «хирургия». И нам необходимо определить соответствующий термин. *Политический процесс* – это действия, с помощью которых вы добиваетесь, чтобы некое лицо или группа лиц выполнили то, чего вы от них хотите. Но эта краткая формулировка охватывает гораздо более сложный круг понятий.

Следствием политического процесса является принятие двумя или более людьми единого курса действий. Обычно у каждого человека, участвующего в политическом процессе, есть личный план, не совпадающий с общим планом группы или личными планами других участников. Именно в результате политического процесса этот личный план становится общеизвестным (в условиях свободы слова), обсуждается и принимается путем голосования (в условиях демократии) либо реализуется без каких-либо дискуссий (если авторство плана принадлежит монарху). В любом слу-

чае участник политического процесса стремится к тому, чтобы группа приняла его план.¹

Кстати, лидеры и менеджеры эту задачу и решают – они добиваются от других людей, чтобы они сделали то, что им требуется. Но вы, вероятно, и сами это уже заметили.

Обратите также внимание, что можно пытаться убедить людей сделать как хорошее, так и плохое дело, но к нашему обсуждению это не имеет отношения. Вы, скорее всего, уверены, что стараетесь заставить их сделать доброе дело, но это, разумеется, ваша личная позиция. Сколь бы ни были оправданны ваши цели, кто-то другой вполне может считать их ложными, дурными, ошибочными и даже злонамеренными.

Три сценария

Если же вы из тех, кто находит, что не создан заниматься политикой, попробуйте взглянуть на это дело с другой стороны. Достаточно рассмотреть всего три случая.

1. Вы король

Может показаться, что в этом случае политика вообще не нужна. Упрощенно можно представить себе, что деспот, даже если он благодушен, заставляет вассалов выполнять свою волю, что они и делают беспрекословно. Если появляются какие-то политики, вы подвергаете их удушению, несогласные рабы кончают жизнь в потайных подземных темницах,² и заниматься какой-либо политикой не надо. В бизнесе эквивалентом является увольнение всякого, кто проявит политические интересы.

Но это чрезмерное упрощение. Даже королям приходится обеспечивать себе поддержку среди ближайших советников, чтобы сохранить за собой власть. Неспроста сегодня осталось так мало неограниченных абсолютных монархий. Если вы действительно самодержец и стремитесь к полному контролю в своем царстве, то можно на какое-то время отстранить от себя политику. Просто в современном мире такая стратегия не может быть разумной, жизнеспособной и долгосрочной.

¹ Майк Перроу (Mike Perrow), мой редактор в «The Rational Edge», блестяще перефразировал здесь мою менее понятную формулировку.

² Так называемых «oubliette» – жутких местах в подвалах замков, название которых происходит от французского глагола *oublier*, «забыть».

2. Вы не король

Тогда у вас два варианта: убедить короля сделать то, что вам нужно, либо объединиться с другими вассалами, чтобы поспорить с королем или сбросить его с трона. Отмечу, что обе альтернативы предполагают наличие политического процесса. Определение политического процесса можно найти чуть выше.

3. Короля нет

Это бывает в организациях, где отсутствуют сильные лидеры. Можете даже назвать это зарождением демократии. В такой ситуации люди должны собраться и решить, что делать, потому что никто не придет и не даст им указания. Обычно люди придерживаются разных мнений, поэтому мы снова вынуждены принимать решение путем достижения согласия (тоже политический процесс).

Можно соглашаться или нет с тем, что технические организации представляют собой «демократии» (все мы знаем менеджеров, которые с этим решительно не согласны!), но важно помнить, что мы существуем в рамках культуры, в которой весьма поощряется активность всех ее членов. На практике самыми сильными оказываются такие компании, которые объединяют таланты всех своих членов, поощряя свободный обмен идеями. Конечно, было бы желательно создать среду, благоприятную в смысле доверия¹ — такую, где существует высокая степень доверия между работниками всех уровней, где царит интеллектуальная честность и где существует минимум политики. Но мы только что показали, что без политики не обойтись ни в одной организации, и технологические компании не исключение. Как же справиться с этим очевидным парадоксом?

Политика неизбежна, но...

Почти в каждой организации в какой-то мере присутствует политика. Для деятельности Палаты представителей и Сената политический контекст и связанные с ним торговые сделки являются основополагающими. Но, как я уже отмечал, когда дело касается технологических компаний,

¹ Это очень важное понятие, к которому я еще вернусь в этой главе. Атмосфера высокого доверия ценна в любой организации, но я пришел к выводу, что в технических организациях она имеет особо важное значение.

большинство их членов хотели бы свести политический контекст к минимуму. Почему?

Ответ прост. Технические проблемы невозможно решить путем политического компромисса. Если для решения проблемы предлагаются два подхода и с технической точки зрения один из них осуществим, а другой вызывает смех, то никакой политический компромисс здесь невозможен. Представьте себе такую абсурдную ситуацию: одна «партия» предлагает построить сплошную плотину, а другая – перфорированную, наподобие швейцарского сыра. Политический компромисс, при котором в плотине оставляют одну или две бреши, не имеет никакого смысла.

Большинство людей сразу понимает, о каких проблемах говорится в предыдущем абзаце, но так бывает не всегда. Мне приходилось встречать разумных во всех других отношениях людей, которые наивно пытались выполнять роль посредников между компетентными и некомпетентными людьми, придавая одинаковую важность идеям тех и других. Это бесполезное занятие, которое раздражает обе стороны и может удручающе действовать на действительно компетентных людей. Да, можно заявлять, что «каждый имеет право на свое мнение», но технические специалисты знают, что не все мнения одинаково компетентны.

Другим, более тонким, вариантом политической ловушки для технических организаций является «*бартер*». Часто в политическом процессе стороны заключают сделку: вы уступаете в одном, а я – в другом. Каждый одно очко выигрывает, а другое проигрывает. Этот способ выхода из тупика встречается чаще всего.

Должен сказать, что в технологических компаниях такой способ часто тоже оказывается неприемлемым. Если требуется соорудить две плотины, то опять-таки бессмысленно строить плотину А согласно философии «цельности», а плотину В – руководствуясь философией «дырчатости». Одна плотина будет держать воду, а другая нет. Вот и все.

Ошибочен также вариант, в котором заблуждающаяся сторона получает право осуществлять свои идеи в какой-то другой области. Например, если решение о строительстве сплошных плотин принимается ценой политического компромисса, в результате которого «дырочникам» отдается на откуп создание каких-то других сооружений, например крепостей, то это плохое решение, ставшее результатом «политических игр».

Конечно, выбор, который приходится делать на практике, не всегда столь очевиден. Если бы всегда приходилось выбирать лишь между «сплошными» и «дырявыми» плотинами, жизнь была бы намного проще. Часто

провести различие между двумя или более вариантами не так легко, а технические проблемы могут быть очень неясными. В таких ситуациях каждая из сторон бывает не готова допустить, что и у противоположного подхода есть некоторые достоинства. Часто технические вопросы отступают на задний план, и дискуссия вырождается в политическую. Как правило, это скверная ситуация.

Когда обстановка политизируется

Каково же законное место политики в технологической компании? Следует признать, что *чисто* технические задачи должны решаться технически. Как только задача выходит за рамки технической области, она попадает в область политики. К сферам, где вмешательство политики обоснованно, можно отнести маркетинг, удовлетворение запросов клиентов, влияние на бизнес и т. п. Поскольку в этих областях труднее определить объективные технические критерии, то необходимы дискуссии, в том числе политические.¹

Однако политика бывает хорошей и бывает плохой. Политологи, конечно, не согласятся со мной. Для них все методы в каком-то смысле более или менее эффективны, и они не любят делать субъективные оценки. Мое разделение на категории они могут счесть наивным, но я считаю его полезным. Рассмотрим поближе, что такое «хорошая политика» и «плохая политика».

Хорошая политика

Следующие приемы можно считать допустимыми и оправданными в политическом процессе:

- Просвещение
- Убеждение
- Достижение согласия
- Поиск фактов

¹ Стоит упомянуть еще об одном исключении. Нередко технически более грамотное решение может оказаться менее удачным выбором, если в компании очень эффективно применяются инструменты и методы, глубоко проникшие в ее культуру. Хотя с чисто технической точки зрения выгоды перехода на более современный инструментарий могут казаться очевидными, связанные с переходом издержки могут перевесить преимущества. Вот здесь желателен какой-то разумный политический компромисс. Неплохо в этой ситуации попытаться поставить себя на место другой стороны.

- Интеллектуально честные дискуссии
- Определение общих интересов
- Обнародование тонких и скрытых обстоятельств
- Поиск компромисса
- Совместное рассуждение

Подобную политику мог бы поддержать Платон. Замечено, что в организациях, где старательно добиваются консенсуса, решения обычно реализуются более гладко, потому что различные фракции собираются вместе и, приводя свои доводы, вырабатывают общее решение. Такая процедура фактически «рекламирует» идею, и окончательное решение оказывается более привлекательным для участников, чем оно могло быть в отсутствие действий, нацеленных на достижение согласия.

Между прочим, я с гордостью отмечаю, что для ранней культуры Rational Software было очень характерно уделять большое внимание достижению консенсуса с целью выработки хороших решений. Несколько позднее нам пришлось перейти к «поиску соглашения в условиях ограничений по времени», когда при невозможности достичь консенсуса к некоторому заданному моменту решение принималось проще и менее демократично, – чтобы не задерживать работу, это делал менеджер. И даже в этом случае времени, в течение которого происходил поиск соглашения, обычно было достаточно для оглашения всех альтернативных точек зрения.¹

Нейтральная политика

Следующие формы поведения представляют собой спорную область. Кто-то допускает их в качестве законных элементов политического процесса, кто-то питает к ним отвращение. Отнесу их к «нейтральной» категории – это единственное субъективное мнение, которое я себе здесь позволю:²

¹ Надо следить, чтобы поиск соглашения не вырождался в «групповое мышление». Сущность процесса состоит в животворности конкуренции мнений. Если консенсус достигается путем оказания давления на коллег, возможно появление плохих решений.

² Здесь тоже надо сказать, что это справедливо для североамериканской культуры. Часть типов поведения из этой пограничной зоны или даже они все могут считаться нормальными, приемлемыми, сомнительными или же недопустимыми в странах Европы или Азии. Ситуация здесь очень сложная. Но чтобы обсуждение стало вообще возможным, мне потребовалось определить три категории. Их содержание отражает точку зрения, распространенную в Северной Америке, и то с некоторым приближением.

- Лесть
- Насмешка
- Лоббирование
- Проволочки
- Отвлечение внимания
- Позиционирование
- Отказ всегда говорить всю правду
- Ты – мне, я – тебе
- «Жульничество»

«Ты сделаешь для меня это, а я сделаю для тебя то» – в принципе это означает отдать одно очко, чтобы выиграть другое, что в начале этой главы я назвал «бартером». То, что один считает политической сделкой, другой назовет разумным компромиссом. Поэтому такой тип поведения я поместил в пограничную зону.

Плохая политика

Не тратя много времени на нейтральную зону, я перехожу к таким аспектам политического процесса, которые большинство людей считает отталкивающими и «переходящими границы допустимого»:

- Намеренная ложь
- Подкуп
- Запугивание и угрозы
- Подкопы, заговоры, интриги
- Личные выпады, оскорбительное поведение
- Обструкция
- Обсуждение мелких вопросов с целью истощить время и терпение
- Тайные намерения
- Взятие на себя обязательств сделать то, чего в действительности вы делать не намерены
- Взятие на себя обязательств не делать того, что в действительности вы намерены сделать
- Обращение к начальству с целью сорвать процесс
- «Цель оправдывает средства» или «В любви и на войне все средства допустимы»

Иногда такого рода политику называют «макиавеллиевской». Это очень несправедливо по отношению к Маккиавелли, который сказал и много другого – помимо того, что «цель оправдывает средства».¹

Если вы столкнулись с плохой политикой, значит, вы вступили в отношения с людьми, для которых политика – это не просто дополнительное средство решения задач. Вы имеете дело с людьми, для которых политический процесс сам по себе является главным занятием. Этим людям важнее победить (с помощью политики или иных средств), чем выполнить свою задачу. У них извращенные приоритеты, и потому они наносят вред организации. Если вы добрались до последней стадии, которая в сущности означает «в поножовщине нет никаких правил», значит, вы зашли слишком далеко.

Соответствие между представлениями инженеров и обычными представлениями

Проблема пограничной зоны в том, что многие технические специалисты тяготеют к самому верхнему краю спектра честности. Поведение из промежуточной зоны они часто относят к категории плохой политики. Например, отказ всегда говорить всю правду формально означает ложь, тем не менее одни допускают здесь разные оттенки, а другие видят только два цвета – черный и белый. Трудно создать среду с высокой степенью доверия, когда поведение слишком часто попадает в пограничную зону. Поэтому не удивляйтесь, если в разговоре с инженерами и техническими менеджерами обнаружите, что они сваливают «нейтральную (пограничную) зону» и «плохую» политику в одну кучу, презрительно называя ее «политикой», а категорию «хорошей» политики характеризуют как «лидерство» или «правильное управление». При этом весь мир становится черно-белым, и всякая политика в нем оказывается злом. Поскольку с этого я и начал, то хочу теперь пояснить, что со всем этим можно разобраться, и остается лишь проблема эквивалентов в языке. При этом может оказаться полезной табл. 13.1.

Можно сказать, что вся эта глава направлена на то, чтобы примирить «общеупотребительное значение» с «инженерным соответствием». Это два разных мировоззрения, но если не разобраться в различных смыслах, ко-

¹ Приношу также извинения Сунь Цзы, которого мне вообще не удалось пристроить в этой главе.

торые придаются словам, могут возникнуть большие неприятности. Например, когда эти инженеры решат, что мы защищаем политику.¹

Таблица. 13.1. Соответствие политик

	С точки зрения политолога	Общепотребительное использование	Для инженеров и технических менеджеров
«Хорошая политика»	Все входит в политический процесс	Допустима	«Лидерство»
«Нейтральная политика»		Сомнительна	Неприемлема
«Плохая политика»		Неприемлема	Неприемлема

Среда с высокой степенью доверия

О среде с высокой степенью доверия я упомянул в контексте культуры, свойственной организации. Вероятно, следует подробнее об этом рассказать и объяснить, почему такая среда столь желательна.

В среде с высокой степенью доверия приняты следующие нормы:

- Можно рассчитывать услышать друг от друга правду;
- Можно рассчитывать на помощь коллег в любом деле;
- Можно рассчитывать на то, что задачи всей организации каждый ставит выше личных или групповых задач;
- Предполагается интеллектуальная честность во всех дискуссиях;
- Предполагается ответственный подход к взятым на себя обязательствам: невыполнение обязательств рассматривается как злоупотребление доверием. Это предполагает наличие как доброй воли, так и компетентности.

¹ Часто система ценностей инженеров дает о себе знать любопытным и противоречивым образом. Например, они, как правило, испытывают дискомфорт, общаясь с клиентами, если считают, что не все, что те говорят, – чистая правда. Здесь они руководствуются стандартами, которые иногда оказываются слишком строгими. Но их очень удивляет, когда их собственная честность ставится под сомнение, если они не сумели уложиться в график в какой-то контрольной точке. Для них это всего лишь техническая проблема, а не нарушение обязательств или злоупотребление доверием. Работая с технологическими компаниями, важно понимать такие особенности, потому что очевидные разногласия с другими подразделениями могут затруднить общение и обмен информацией.

Замечательное свойство среды с высоким доверием – ее чрезвычайная эффективность. Если есть доверие, меньше надобности в проверках. В результате меньшая команда может выполнить больше работы. Это все равно, что уменьшить трение в механизме: больше энергии пойдет на совершение работы и меньше – на выработку тепла. Обычно высокого доверия проще достичь в небольших и относительно однородных группах. По мере укрупнения команд, роста их распыленности и функциональности поддерживать атмосферу высокого доверия становится все труднее. По иронии судьбы потребность в ней не уменьшается, а сложность растет.

Атмосфера высокого доверия благотворна в любых организациях. И хотя в технологических компаниях она так же важна, как во всяких других (если только не более), установить ее там еще труднее. Если говорить о мировоззрении инженеров (см. табл. 13.1), то видно, что даже «нейтральная» политика отравляет атмосферу высокого доверия. Поэтому для создания в организации атмосферы высокого доверия придется повысить требования и запретить больше типов поведения из «пограничной» зоны, чем было бы допустимо в другой ситуации.

После работы в атмосфере высокого доверия испытываешь трудности, оказавшись в атмосфере с более низким доверием. Все инстинкты оказываются ложными, и доверчивый человек в среде с низкой степенью доверия легко становится жертвой политических махинаций и может пострадать. И наоборот, тем, кто пришел из атмосферы с низкой степенью доверия, важно, попав в среду с высоким доверием, научиться доверять другим: паранойю, поддерживавшуюся долгие годы, бывает трудно излечить. Эмигранты из бывшего Советского Союза рассказывали мне, что приспособление к американской культуре, которой в меньшей степени свойственно состояние «повышенной боевой готовности», представляло для них известную сложность.

Другие разновидности плохой политики

Помимо тех типов поведения, которые перечислены в разделе «плохая политика», могут встретиться и другие отрицательные типы. Я привожу их здесь, чтобы они не прошли мимо вашего внимания.

Существует такая особенно вредная форма политики, как нежелание «подчиниться». В каждом процессе за периодом обсуждения следует принятие решения. После этого «избирательные участки закрыты». После того

как решение принято, все должны ему подчиниться.¹ Продолжать хныкать и агитировать за отклоненное решение после состоявшегося факта – плохая политика. Вместо того чтобы дать организации возможность двинуться вперед и реализовать решение, такая политика погружает ее в трясину обсуждения. Хуже того, отказ подчиниться подрывает структуру руководства, снова и снова оспаривая решения. Этот вид плохой политики следует безжалостно искоренять. Менеджеры должны со всей силой дать понять, что несогласным придется покинуть команду. Для здорового политического процесса это разумное условие.

Другой симптом плохого политического поведения – явное строительство империи. По-прежнему очень многих привлекают атрибуты и символы успеха: положение в организации, количество непосредственно и косвенно подчиненных сотрудников, объем финансирования и т. д. Организации, придающие чрезмерное значение таким статистическим символам, пожинают плоды своих трудов: поведение в них определяется системой вознаграждения. Если же, напротив, ваша организация поощряет оригинальность, вклад сотрудников, упорство в решении задач и т. д. независимо от должностного положения, то вы на правильном пути. В таких организациях прежний стиль поведения выглядит болезненным пережитком и может осуждаться господствующей культурой. Тот, кто упорствует в нем, просто относится к другой культуре – более политизированной и придающей вес мнениям в зависимости от места их автора в иерархии, а не от подлинной ценности. В организациях с атмосферой высокого доверия такое поведение в конечном счете должно быть отвергнуто, и если оно не изменится, то отвергнуты должны быть его носители, и чем скорее, тем лучше.

Наконец, существует проблема стиля, часто относимая к политическому поведению обычно негативного характера. Есть люди, любое взаимодействие с которыми превращается в преодоление препятствий: они всегда «напрягают» окружающих. Такие люди, как минимум, становятся обременительными и начинают отнимать силы у коллег: все уже понимают, что в общении с ними всегда приходится занимать «оборонительную позицию». Если я попытаюсь привести вам здесь соответствующий пример, меня точно привлекут к суду за клевету.² Но все поймут, о чем я здесь говорю,

¹ Я посвящаю вопросу подчинения решению всю главу 15.

² Аналогия из области спорта: в теннисе такого рода поведение родственно тому, что партнер постоянно выходит к сетке. На противника это оказывает непрерывное давление.

потому что в любой организации найдутся люди, которые вызывают раздражение именно таким поведением. С ними просто не хочется иметь дело. Постарайтесь сами не быть такими и удерживать других от такого поведения.

Резюме

Политический процесс – это часть поведения человека и организации. Попытаться отстраниться от него – значит уступить позиции тем, кто захочет участвовать в этой игре. С точки зрения морали и логики несостоятельно сначала избегать политики из чувства отвращения, а потом жаловаться, что вас лишили всех политических прав.

Однако в качестве руководителей мы обязаны препятствовать вмешательству политики в чисто технические проблемы. Даже на государственном уровне мы иногда ошибочно позволяли политическим соображениям доминировать при принятии решений, неприемлемых в техническом отношении. Когда факты очевидны, нельзя игнорировать их, каковы бы ни были политические последствия. Не пытайтесь бороться с законами физики.

Если мы занялись политикой, то должны поощрять хорошую политику, как она обозначена в этой главе. Заметив признаки поведения, относящегося к «нейтральной зоне», надо проявить осторожность. А плохую политику необходимо искоренять и душить, предпочтительно с помощью давления, оказываемого коллегами.¹ Обычно плохая политика уродует весь процесс, допуская явно неэтичное и бесчестное поведение.

Обратите внимание, что даже действия из «нейтральной» или «пограничной» зон разрушают атмосферу высокого доверия. Преимущества атмосферы высокого доверия настолько высоки, что следует серьезно взвесить, стоит ли рисковать ей, разрешая приемы из «пограничной зоны».

Необходимо обеспечить, чтобы все политические процессы и переговоры завершались после «закрытия избирательных участков», чтобы принималось решение, команда подчинялась этому решению и все как один брались за его реализацию, прекратив всякие непроизводительные обсуждения.

Уже было сказано, что зло торжествует, когда хорошие люди ничего не делают. Если вы прибегнете к тому аргументу, что «все политики – корруп-

¹ Это пример хорошего давления со стороны коллег.

пированные негодяи», то я произнесу только два слова: Гарри Трумен. Трумен не разбирался в технике, но он был превосходным политиком и лидером.¹ Эта тема увела бы нас далеко за рамки данной главы, да и всей книги в целом. Но у меня постоянно вызывает беспокойство (полагаю, что обоснованное) то обстоятельство, что из-за нежелания разработчиков программ участвовать в процессе, рассматриваемом ими как политический, организация терпит значительный ущерб.

Но политические проблемы возникают не всегда. Многие разработчики предпочитают не участвовать в них, поскольку в прошлом мы осуждали их участие. Многие опытные люди отмечают, что менеджеры все равно ничего не слушают. Возможно, такая позиция несколько предвзята, но если мы требуем участия, то должны прислушиваться. И, разумеется, всегда необходимо отделять технические проблемы от нетехнических.

Разделить технические и нетехнические проблемы иногда очень просто. Как правило, *что* делать – это политическое решение, а *как* – техническое. Однако они могут быть взаимосвязаны. Например, время и ресурсы, необходимые для реализации некоторой функции (*как*), могут повлиять на решение о том, заниматься ей или нет, т. е. *как* оказывает влияние на *что*. Здесь важно иметь уверенность в том, что представленная инженерами оценка продолжительности работы сделана честно, а не под влиянием мыслей о возможных политических последствиях. В последнем случае вам грозят неприятности.

Но иногда даже умеющие слушать менеджеры сталкиваются с трудностями, как только пытаются сами открыть рот. О том, как разговаривать с инженерами, повествует следующая глава. Я озаглавил ее «Ведение переговоров», поскольку так этот процесс представляется обеим сторонам. Глава посвящена общению между людьми, занятыми разработкой ПО, в целом.

¹ По моему мнению, Трумен был человеком исключительной честности, хотя он и был выходцем из крайне коррумпированной в то время политической организации в Америке – партийной машины Пендергаста в Канзас-сити. Меня всегда интересовало это несоответствие.

ГЛАВА ЧЕТЫРНАДЦАТАЯ

Ведение переговоров

Почему иногда не получаются даже простейшие вещи?

Я полагаю, что первопричиной таких ситуаций, как правило, является классическое «взаимонепонимание». «Как, разве вам это было нужно? Почему же вы не сказали?» Мы раздражены, мы пытаемся справиться с отчаянием. Но пока мы не разберемся, в чем здесь неправильность, мы обречены бесконечно повторять свой печальный опыт.

Интересно отметить, что такое взаимонепонимание редко возникает, если проведены переговоры. После взаимных уступок и выработки решения вероятность серьезного конфликта значительно уменьшается. Может быть, в этом и заключается главное. Весь фокус, возможно, в том, чтобы провести переговоры без собственно переговоров. Потому что, как всем нам известно, переговоры чреваты конфликтами, и совсем нежелательно, чтобы любой контакт приобретал такой характер.

В этой главе наш друг Роско Леруа излагает свой план.¹

Общение – это все

Однажды мы с Роско сидели около печки и брюзжали на субботний дождь. Роско уже примерно полгода вел проект с участием некой группы разработчиков ПО, и, к моему удивлению, при этом не произошло

¹ С Роско мы уже встречались в главах 5, 10 и 11. Можете обратиться к ним, чтобы выяснить, что это за фигура.

никаких событий, достойных отметки на шкале Рихтера. Если учесть, что Роско взрывоопасен, как Везувий в августе,¹ я предположил, что что-то назревает.

– Роско, – решил я, – как это ты до сих пор не разогнал половину команды?

– А зачем? – отвечал он. – Они, знаешь ли, хорошо работают. Если, конечно, уметь с ними разговаривать.

– Да неужели? – удивился я. – А что, разве беседы с программистами – это что-то особое?

– Вот тут ты ошибаешься сынок. Надо не *беседовать* с ними, а *разговаривать*. Начнешь беседы проводить, так и глазом не моргнешь, как начнешь выговоры им делать, а там и до беды недалеко.

Черт! Прижал меня так быстро, что я опомниться не успел. Но он был прав.

– Ладно, Роско, согласен. Но в самом деле, отличается ли общение с инженерами, особенно с программистами, от разговоров с другими людьми? Если да, то мне очень хотелось бы понять, в чем тут секрет.

Роско излагает свою теорию

– Ну, дело вот в чем. Для начала надо отметить, что разработчики ПО хотят, чтобы к ним относились как к профессионалам. Они не просто нуждаются в уважении, как все люди, но требуют его, как доктора и другие профессионалы. Если считать их просто «программерами» или «хакерами», они разозлятся. Поэтому важно вести себя «с пониманием».

– Черт возьми, – высказался я, – ты говоришь так, будто речь идет об обращении с *примадоннами*.

– Не совсем так, – сказал он, и это удивило меня. – Пойми, разработка программ – это самая молодая из инженерных дисциплин. Всем известно, чем занимаются инженеры в машиностроении, строительстве, энергетике и химическом производстве, а в авиации инженеров даже иногда называют учеными-ракетостроителями. Но не так много людей понимает, чем в действительности занимается инженер-программист. Проблема

¹ Извержение вулкана Везувий, расположенного к югу от Неаполя, произошло 24 августа 79 года до н. э., в результате чего города Помпеи и Геркуланум исчезли под лавой, пеплом и грязью.

в том, что программисты часто ощущают себя эдакими Родни Дэйнджерфилдами в инженерии.¹ Поэтому я их слегка одергиваю.

– ОК, – сказал я, – только не рассказывай мне про их комплекс неполноценности. Некоторые из них кажутся мне довольно невежественными.

– Так ты хочешь узнать что-то об общении или нет?

Я был вынужден умолкнуть.² Мне надлежало слушать дальше.

Меня удивило, кстати, что Роско относит разработчиков ПО к «инженерам». Многие из тех, кто занят в софтверной индустрии, считают, что нам еще далеко до того, чтобы стать настоящей инженерной дисциплиной. Но я решил не дразнить гусей. Менеджерам приходится общаться с теми, кто проектирует и пишет наши программы, независимо от того, как мы именуем этих людей. И я знал, что если мне потребуется мнение Роско по этому предмету, он его выскажет. И, черт побери, если ему потребуется мое мнение, то он его тоже выскажет!

Четыре шага

– Метод общения с инженерами-программистами есть, и он состоит из четырех шагов. Вроде техасского тустепа, повторенного дважды.

Роско готовился к выступлению. Он несколько раз пыхнул сигарой и налил себе еще кофе.

– Если сделать эти четыре шага, то, скорее всего, тебе удастся спасти свою шкуру. Если же ты рискнешь какие-то из них пропустить, то от тебя живого места не останется. Так что слушай внимательно.

Шаг первый

– Во-первых, запомни, что имеешь дело с инженерами. Инженеры обычно не любят вести светские беседы: они считают это бесполезной тратой своего времени. А время свое они ценят дорого. Поэтому прежде всего они хотят знать, в чем заключается задача.

– Ты хочешь сказать, что каждый разговор должен быть построен вокруг какой-то конкретной задачи? – спросил я. – А если я хочу обсудить с инженером перспективы?

¹ Для наших заокеанских читателей: Родни Дэйнджерфилд (Rodney Dangerfield) был американским комиком, коронная фраза которого звучала так: «Меня никто не уважает!»

² Я вспомнил, как возмущался, что моя теща кладет сахар в горох, и как она мне ответила: «Так ты хочешь, чтобы дети ели его, или нет?»

– Ну да, проблемы, перспективы – как вы, менеджеры, любите их называть. Ладно, назовем это *вопросом*, если тебе так больше нравится. Суть в том, что работа инженера заключается в решении задач, и если ты пришел с ним поговорить, то он, естественно, полагает, что у тебя есть задача, которую надо решить. Поэтому обычно я сразу просто говорю, что есть задача, которую я хотел бы обсудить, а потом просто говорю, в чем она состоит.

Мне показалось, что задание такой начальной точки отсчета для разговора может быть неплохой идеей. Чем ходить вокруг да около, лучше сразу выложить, зачем ты пришел. Поэтому я кивнул и стал ждать продолжения.

– Иногда он недовольно ворчит в ответ и быстро объясняет, почему твоя задача вовсе не является задачей. Иногда оказывается, что твоя информация была неверна. А иногда, что задача поставлена некорректно. Вполне уместно проявить настойчивость. Если сумеешь более четко описать задачу, то заслужишь некоторое уважение инженера. Не забывай, что инженер старается оценить как тебя самого, так и твою задачу. Он ни секунды не потратит на задачу, которую считает никчемной или несостоящей. Если у вас разногласия, лучше обсудить их, потому что пока он не будет убежден в том, что есть достойная задача, ты ничего от него не получишь.

Я вспомнил несколько дискуссий, во время которых я преждевременно забегал вперед. Естественно, каждый раз приходилось возвращаться и снова вести разговор, пока мы не достигали согласия относительно сущности проблемы. Есть у инженеров такая интересная особенность.

Шаг второй

– Теперь, когда вы оба согласились, что проблема существует, необходимо определить права собственности, – продолжил Роско.

Права собственности? Роско явно успешно осваивал профессиональный жаргон.

– Дело вот в чем. Если это *твоя* проблема, тогда зачем ты пришел? Услышать совет или рекомендацию? Ты можешь их получить. Если же ты имел в виду, что это *его* проблема, ну, тогда ему, возможно, придется что-то сделать. Поэтому инженер, прежде чем обсуждать задачу дальше, хочет знать, *чья это проблема?*

– Есть еще одна возможность, – сказал я. – Это может быть проблема моя и не его, а кого-то постороннего.

– Да, возможно. Но тогда можешь быть уверен, что инженер скажет тебе что-то вроде: «А чего ты ко мне с этим пришел, если это не твоя проблема»

и не моя? Пусть он (посторонний) об этом и беспокоится.» Поэтому тебе придется объяснять, что это и к нему имеет отношение. Иначе он сочтет задачу неинтересной или не стоящей его времени.

Теперь я видел, что все не так просто, как мне казалось.

– Если это твоя проблема, то в лучшем случае можешь рассчитывать на внимательный анализ, возможно, рекомендацию, после чего лучше убраться из Доджа.¹ Инженер очень неохотно будет тратить *свое* время на *твою* проблему. Ну, вот такие они люди.

– С другой стороны, если ты считаешь, что это *его* проблема, придется приложить некоторые усилия, чтобы он признал это. Если инженер принимает на себя ответственность за проблему, он начинает действовать профессионально, чтобы решить ее. Поэтому инженеры неохотно встречают всякую новую задачу, которую ты им предъявляешь. Просто удивительно, на что они готовы пойти, лишь бы не признаться, что проблема касается их. Иногда они могут даже отрицать существование такой проблемы в принципе.

– Ага! – воскликнул я. – Вот почему тебе сначала понадобился шаг первый! Необходимо закрепить существование проблемы, чтобы она не улетучилась в процессе последующего обсуждения.

– Ты ухватил суть, – сказал Роско, – да, ухватил.

Итак, отметил я, вначале все просто. Устанавливаешь существование проблемы и убеждаешь инженера, что это *его* проблема. Первый тур тexasского тустепа дался с легкостью.

– Окей, – сказал я, – как насчет второго тура?

Шаг третий

– Ну, чаще всего именно после этого все летит под откос, – сказал Роско. – Самое главное теперь *не* предлагать решение.

Черт меня подери, подумал я. Что такого ужасного в том, чтобы предложить решение?

– Дело в том, что многие инженеры-программисты ошестиниваются при слове «требования». В прежние времена к ним приходили с готовыми

¹ Тоже для заокеанских читателей. Додж-сити – это город, который находился на западном рубеже земель, занятых пионерами, и был известен широким применением насилия и «приграничного правосудия». Считалось, что новичкам в нем лучше не задерживаться. Поэтому выражение «убраться из Доджа» получило смысл «не болтаться слишком долго в опасном месте».

идеями относительно способа реализации, называвшимися «требованиями» (requirements). Такая практика приводила к множеству ненужных конфликтов. Инженер желает знать, *что* должно быть сделано. Он не желает слышать от тебя, *как* это должно быть сделано. В конце концов, это то, за что ему платят деньги. Если ты бодро подходишь к нему с готовым решением, почти наверняка у вас возникнут разногласия. Есть и другая проблема, которая возникает, если предлагается готовое решение. Часто инженер, исходя из твоего «решения», пытается выяснить ход твоих мыслей и определить, что тебе нужно *в действительности*. При этом он может ошибиться. Поэтому, предлагая решение, ты создаешь дополнительный объем работы и подвергаешь риску все мероприятие.

– В таком случае, – стал думать я вслух, – откуда же берется решение?

– По-настоящему толковые менеджеры, – продолжал Роско, – умеют «прощупать» своих инженеров и выяснить диапазон возможных решений. Они заводят разговор о возможных вариантах, их относительных выгодах и недостатках. Нет ничего плохого в том, чтобы изучить «пространство решений», но основным докладчиком должен быть инженер. И ради бога, не учи его держать карандаш в руке. В программировании практически всегда можно решить задачу массой способов. Инженеры тем и занимаются, что рассматривают альтернативные варианты, а потом пытаются выбрать из них лучший – то, что они называют *оптимальным* решением.

– Минуточку, – вмешался я, – а кто будет решать, какое решение лучшее?

– Хорошее замечание, – ответил Роско. – «Лучшее» может означать то, которое «быстрее», «дешевле», «сопряжено с меньшим риском», «требует меньше сопровождения», «более искусно», «более элегантно», чем все другие, или выделяется еще какими-то достоинствами. Поэтому во время изучения пространства решений вместе с инженером нужно разобраться, что означает «лучшее» в твоем бизнес-контексте. Иначе можно получить технически «лучшее» решение, которое непригодно для твоих потребностей.

– Хорошо, – парировал я, – а если реализация решения требует слишком много времени?

– Тоже верное замечание. Цель изучения пространства решений вместе с инженером состоит также в том, чтобы дать ему представление о том, какие *ограничения* ты вынужден наложить. При этом вы можете договориться о решении, которое каждого из вас удовлетворит лишь отчасти, но иногда это лучшее, чего можно достичь.

Так, посмотрим, что у нас получилось. Мы описали задачу, мы определили владельца, наметили пространство решений и обсудили некоторые альтернативы. Неплохо. Что осталось?

Роско, конечно, читал мои мысли.

Шаг четвертый

– Подведение итогов, – сказал Роско. – Когда имеешь дело с инженерами, всегда надо подвести итоги. Например, «какие будут следующие действия?» и «кто и что должен сделать и к какому сроку?» Если обсуждение заканчивается, а эти темы не затронуты, то результат почти всегда оказывается печальным.

– Почему? – поинтересовался я.

– Дело обстоит примерно так. Инженеры люди занятые. Задач у них всегда больше, чем времени на их решение. Часть этого времени ты у них уже отнял своими шагами с первого по третий. И оно будет потрачено впустую, потому что без шага четвертого все на том и закончится, как только ты выйдешь из комнаты.

– Как это? – вырвалось у меня.

– Ну посмотри: ты дал инженеру указание что-то сделать? Ты дал распоряжение, чтобы он действительно начал работу? Ты определил для новой задачи очередность относительно тех, над которыми он уже работает? Нет. Ничего этого ты не сделал. Поэтому ясно, что ты приходил, чтобы поговорить о задаче, а не чтобы ее решить. Если ты хочешь, чтобы инженер действительно что-то сделал, ты должен *сказать* об этом ясно и четко. Это надо выразить явно и недвусмысленно. И если он берет на себя обязательство что-то сделать, то, конечно, надо предложить ему установить срок. Потому что без срока это обязательство несерьезное. Далее, чтобы действительно завершить шаг четвертый, могут потребоваться еще некоторые действия. Инженер может потребовать дополнительные данные, попросить точнее описать объем работы и даже, возможно, представить это все ему в письменном виде. И ему может понадобиться еще некоторое время на размышление, после чего вы окончательно договоритесь по поводу реализуемого решения. Все это разумные требования. По крайней мере, ты теперь знаешь, что он занимается этой задачей, и можешь рассчитывать на результат.

Смотрим глубже

Итак, рецепт Роско довольно прост:

1. Описать задачу и согласовать ее описание с инженером.
2. Установить за инженером права собственности на задачу.
3. Исследовать пространство решений под руководством инженера.
4. Явно сформулировать и согласовать решаемую задачу, включая ее объем, приоритет и срок выполнения.

– Почему же, – спросил я у Роско, – этим простым рецептом не пользуются чаще?

– Думаю, здесь играет роль то, что инженеры называют *несоответствием импедансов*.¹ Менеджеры многое считают само собой разумеющимся и делают массу предположений, которые попросту неверны. Иногда у них отсутствует правильная информация, часто они просто не знают, чего они не знают. Но поскольку они сами умные и знают, что инженеры тоже умные, им кажется, что существует общий контекст для обсуждения. Поэтому они забегают вперед. Часто они сразу перескакивают к решению, выполняя шаги третий и четвертый.² А инженер все еще мысленно находится на первых двух шагах. И дело не движется.

– Иногда можно сберечь время, если двигаться медленнее, – изрек Роско. – И, конечно, есть еще одна проблема. Часто решение, которое предлагает менеджер, сляпано на скорую руку, лишь бы срочно решить проблему. Это может быть дешевым выходом на короткое время. Вот тут инженер может разозлиться.

– Но почему? – спросил я.

– Ну, таковы они по натуре. Инженеры ненавидят делать одноразовые продукты. Уж лучше он потратит побольше времени, но сотворит надежное решение, которое не придется переделывать. Иногда вместо внимательного анализа инженерных компромиссов получаются просто заумные разговоры. Но для обсуждения таких вопросов с инженерами менеджер должен быть действительно хорошо подкован технически. Поэтому

¹ У этого термина интересная история. Он взят из электротехники, где означает попытку соединить две цепи, у которых разные возможности поглощения или передачи энергии. При этом получается общая цепь, в которой очень велики потери. В обыденной речи этим термином стали обозначать такое общение между двумя сторонами, в котором их контексты (или языки, на которых они пытаются изъясняться) настолько различны, что общение затруднено и иногда совершенно прерывается. Можно сказать иначе: это процесс, в котором выделяется больше тепла, чем света.

² Забавно, что такую ошибку часто совершают менеджеры, когда-то бывшие инженерами. А уж они-то должны в этом разбираться.

лучше не лезть со своим решением, а дать возможность инженеру предложить свои варианты.

Это пахнет политикой. Интересно, нет ли у Роско других предложений?

– И еще одно. Следи за тем, чтобы не давать инженерам-программистам слишком много заданий одновременно. Они считают, что лучше всего работают, когда концентрируются на одной задаче, отдаваясь ей полностью.¹ Им известно, что такое многозадачность, но они не считают такой режим оптимальным для работы. И знаешь, – подмигнул он мне, – если они так считают, возможно, так оно и есть. Поэтому постарайся не проговориться, что у тебя хватит задач, чтобы завалить их работой выше крыши. От этого у них сразу портится настроение.

Довольно проницательно. Что там у нас еще в запасе?

– В этих делах нужно быть деликатным. Помни, что твоя задача – налаживать общение. Если твоему партнеру по диалогу покажется, что ты его поучаешь или пытаешься им манипулировать, пиши пропало. Общение с инженерами – это процесс, а не разовое действие. Диалог должен быть непрерывным. Нельзя раз поговорить с инженером, а потом забыть про него. Уже это одно свидетельствует о том, что нельзя оказывать на него слишком большое давление. Ты должен разговаривать с ним и дальше, если не на эту тему, то на другую.

– Поэтому, – сказал Роско, – я стараюсь установить такой порядок, когда парень знает, чего ждать дальше. Я рассматриваю это как организацию зоны комфорта. Если он знает, что сначала я постараюсь определить проблему и установить, в чьих интересах ее решение, он может подготовиться к соответствующему обсуждению. Может быть, нам хватит для этого пяти минут. И оба мы сэкономим время. Когда этот этап пройден, может состояться более интересный, с его точки зрения, разговор. Это уже шаг третий, на котором мы обсудим альтернативные решения. Тут уже он может блеснуть. И, конечно, он должен знать, что неизбежен шаг четвертый. Это покажет серьезность моих намерений, что я хочу не просто поговорить, а мне нужны какие-то действия. Повторюсь, что инженеры это уважают. Между прочим, если инженер сам придет к тебе поговорить, можешь предполо-

¹ Говорят, что есть исследования, подтверждающие данную позицию. Хотя Роско не смог сослаться на какое-нибудь из них. Если уж быть совсем справедливым, то нужно заметить, что есть инженеры, которые любят работать над несколькими проблемами одновременно. Правда, такие мне встречались в меньшинстве, отсюда и это обобщение.

жить, что он будет действовать таким же четырехшаговым методом. Это потому, что инженеры примерно таким способом общаются *друг с другом*.

Ну, теперь все?

Осталась еще одна вещь, которая тревожила меня в подходе, предложенном Роско.

– Роско, – сказал я, – мне кажется, что твой метод четырех шагов последователен по своей сути. Я знаю, что при итеративной разработке обсуждений должно быть много, как ты это уже отмечал. Но разве каждый разговор сам по себе не является «итеративным»?

Роско улыбнулся.

– Ты, кажется, сразу хочешь стать «опытным участником», – сказал он. – Было бы здорово, если бы я смог побольше людей заставить проходить эти четыре шага. Это был бы прогресс. Но ты прав, сынок. Иногда, добравшись до шага четвертого, приходится вернуться на третий шаг. Это происходит из-за того, что на четвертом шаге в реализации могут обнаружиться такие детали, что придется заново оценить решение, принятое на шаге три.

– Хорошо, – согласился я, – а не может при этом потребоваться вернуться еще дальше?

– Разумеется, – ответил Роско. – Нетрудно заметить, что иногда на третьем шаге можно обнаружить альтернативное решение, которое потребует участия в работе кого-то еще. Это значит, что надо вернуться на второй шаг, чтобы найти нового собственника и убедить его.

– Значит, это все-таки итеративный подход, потому что я могу представить себе и такую ситуацию, когда придется вернуться вообще на первый шаг и заново сформулировать задачу, проведя обсуждение с третьей стороной, – высказал я предположение.

– Тут необходима осторожность, – сказал Роско. – Ты все правильно говоришь, но цель данного упражнения в том, чтобы подвести конечную черту, а не заниматься итерациями вечно. Некоторое количество итераций полезно, но твоя задача всегда в том, чтобы добраться до шага четвертого и выйти из него с планом действий. В противном случае не исключено, что тебе придется возносить молитвы у алтаря Владычицы Вечных Ревизий.¹

Эту ловушку я тоже заметил. Роско убедил меня в необходимости изменить стиль общения с инженерами-программистами. Даже если придется

¹ Святая-покровительница тонущих проектов.

разговаривать с целой командой инженеров, я буду придерживаться тех же самых шагов: для перехода от разговора «один на один» к разговору «один со многими» потребуется еще больше организованности и дисциплины. Благодаря рецепту Роско группа должна с еще большей очевидностью убедиться в том, что у меня есть вполне организованный способ рассмотрения (и решения) задачи.

Я решил попытаться применить подход Роско и проверить, удастся ли мне развить свои способности к общению.

Резюме

У инженеров есть одна опасная особенность, которую я наблюдал на протяжении многих лет и о которой необходимо помнить. Если описать им некоторую задачу, они всегда стараются рассмотреть ее в самом общем виде. Иными словами, им свойственно предполагать, что вы хотите решить «сразу всю проблему». Иногда это в шутку называют *«преобразованием постоянного тока в дневное освещение»*, потому что в электротехнике ширина спектра частот может изменяться от нуля (постоянный ток) до бесконечности, т. е. далеко за пределами видимого света!

Опасность сопряжена с двумя моментами. Во-первых, решать задачу для всего диапазона возможных значений параметров обычно гораздо труднее, чем для какого-то ограниченного его подмножества. Иногда общее решение вообще невозможно, хотя для какого-то ограниченного диапазона оно вполне осуществимо. Поэтому инженер может в бессилии развести руками, если не сообщить ему, что вас интересует решение только для некоторого конечного диапазона.

Во-вторых, и этот подводный камень еще коварнее, задача может иметь общее решение, а может не иметь, при этом инженер может суметь отыскать это решение, а может не суметь. В худшем варианте он считает, что решение есть, и пытается его найти, что ему не удастся, потому что задача нерешаемая или он недостаточно способный. Это оборачивается большими расходами.

Наконец, есть вариант, когда проблема имеет общее решение и инженер достаточно способный, чтобы найти его. К несчастью, он ищет его слишком долго, и обходится это слишком дорого. Вам нужна лишь малая часть его результатов – одна десятая по объему и одна десятая по стоимости. Он горд собой, а у вас от всего этого остается неприятный осадок.

Единственный способ избежать этой дилеммы состоит в том, чтобы тщательно обсудить сущность задачи на шаге один и шаге три, а также обязательно определить, насколько общее решение вы можете себе позволить, на шаге четыре. Так что у вас есть много путей избежать этой ловушки. Не забывайте, что, как правило, невыгодно строить изгородь за \$50, чтобы охранять лошадь, цена которой – \$10.¹

Не каждый инженер-программист, с которым вам доведется работать, благосклонно воспримет описанный в данной главе протокол, потому что инженеры – живые люди, и у них есть свои особенности. Однако мой опыт показывает, что этот метод жизнеспособен в подавляющем большинстве случаев и потому может послужить хорошей отправной точкой.

В следующей главе мы посмотрим, что следует за шагом четыре. Мы полагаем, что на этом шаге уже заручились согласием инженера сделать что-то к определенному сроку. Наш успех зависит от того, выполнит ли он свое обязательство. Эта базовая парадигма глубже исследуется в главе 15.

¹ Мой старый товарищ Билл Ирвин часто напоминает мне об этом.

ГЛАВА ПЯТНАДЦАТАЯ

Получение согласия

Как правило, планы и графики работ проектов разработки ПО носят агрессивный характер. В редком проекте не встретишь каких-нибудь нововведений или честолюбивых замыслов. Руководство всегда хочет получить результат как можно скорее, а риск является неотъемлемым элементом. При этом большинство разработчиков уже участвовало по крайней мере в одном провалившемся проекте, кое-кто из них был занят в проектах, которые сразу же выпадали из графика и велись мучительно и нескончаемо. Поэтому, как только предлагается некий новый проект, разработчики и менеджеры воспринимают его настороженно: какие сюрпризы ждут их на этот раз?

Напуганная команда не может работать эффективно, поэтому такую изначальную тревогу необходимо побороть. Этого не добиться заказом специальных футболок и устройством вечеринок. Правильный путь заключается в том, чтобы составить разумный план с максимальным участием всей команды, а потом обсудить его в более мелких группах. Не имеет смысла собирать *всю* команду в одном зале и демонстрировать план в виде презентации PowerPoint. С помощью такого шоу вы ничего не добьетесь: они хотят обсудить детали и практические вещи, а это можно сделать только в непринужденной обстановке с небольшим числом участников.

В конце каждого из таких небольших совещаний нужно оглядеть всех сидящих за столом и предложить каждому участнику *подтвердить свое согласие* осуществлять этот план. Недостаточно заручиться согласием руководителя группы; нет, об этом должен заявить каждый из присутствующих. Обсудив план во всех группах и получив одобрение от всех участников, можно обоснованно считать, что вся команда готова к старту. Если

кто-то считает план нереальным, у него была возможность высказать свое мнение. И если этот человек не подтвердил своего согласия, вы должны решить, остается ли он в этой команде.

Но что в действительности имеется в виду, когда идет речь о «подтверждении согласия»? Главное в данном случае – это *принятие обязательств*. В этой главе мы снова познакомимся с точкой зрения Роско.¹

Роско разбивает себе нос...

Опять мы с Роско сидим около печки и брюзжим на субботний дождь. Он уже год работает с группой инженеров-программистов, и у него произошел первый крупный «прокол». Роско столкнулся с неизбежностью, и это меня не удивило. Интереснее было, как он произведет разбор этого всегда неприятного события.

...И сразу переходит к делу

– Итак, – начал Роско, – все очень просто. Ребята меня подвели.

Роско не тот человек, чтобы искать виновных и срывать на ком-нибудь злость. Он не из малодушных. Тем более мне было интересно, что заставило его произнести такие слова.

– Дело было так, – продолжил он. – Они согласились выпустить продукт к определенному сроку, но не сделали этого.

Я не спрашивал, большим ли было опоздание, хотя подозревал, что большим. Что меня заинтриговало, так это «двоичный» образ мышления Роско. Мне показалось, что он занял позицию «все или ничего» и воспринимал только два цвета – белый и черный. Разве окружающая действительность не выглядит лучше в полутонах?

– Вероятно, – сказал я, – были какие-то смягчающие обстоятельства.

Лучше бы я ничего не говорил.

Извержение Везувия

– Чуть собачья! – вскричал Роско. – Послушай меня, сынок. Ни у одного из них руки-ноги не отвалились, начиная с того дня, когда они подписались на этот срок, и кончая тем днем, когда он истек. Какие *к черту смягчающие обстоятельства!*

¹ Если вы все еще не знаете, кто такой Роско, см. главы 5, 10, 11 и 14.

Очевидно, я задел больное место, и содрогнулся при мысли о том, что пришлось пережить команде Роско. Ибо если даже я, сторонний наблюдатель, был поражен его реакцией, то можно только догадываться, как они реагировали на те пинки, которые он раздавал в день предполагавшейся сдачи продукта. Всем известно, что Роско не станет стесняться в выражениях, когда решит высказать людям, что он думает об их работе.

– Пойми, я оставил им большой резерв, – продолжал Роско. – Они сами сделали оценки, а я применил свое правило квадратного корня.¹ И после этого приходят с пустыми руками. Моя первая сухая скважина² за долгое время. Я все время чувствовал, что у них не ладятся дела. Я пытался влезть в их работу и разобраться, что происходит. Но они меня не пускали, говоря, что я не пойму технические проблемы, и все в конечном итоге будет хорошо. Сам виноват, что согласился с ними. Больше я такой ошибки не сделаю.

Роско неохотно признал, что его невмешательство в этом проекте было ошибкой. Я предположил, что он недостаточно глубоко занимался проектом, в отличие от своей обычной практики, поскольку доверился своим людям. Похоже, что в данном случае они оказались недостойны его доверия.

Я поинтересовался у него, на что он рассчитывал? На их безупречность?

Как это делается в Техасе

Тут Роско объяснил мне, что такое *рукопожатие в Техасе*. Мне предстояло расширить свое образование.

– Я вырос на нефтяных полях Техаса,³ – объяснил Роско. – Там, где добывают нефть, мужчина связывает себя словом. Если ты скрепляешь договор рукопожатием, то берешь на себя обязательство. А обязательства священны.

Я попросил его привести какой-нибудь пример.

– Пожалуйста, – сказал он. – Допустим, у меня есть какие-то трубы,⁴ которые мне нужно срочно перебросить на другую буровую. Что я делаю?

¹ См. главу 11 «График работ».

² У нефтедобытчиков «сухая дыра» – это скважина, которая не дала ни капли нефти. Это нулевая окупаемость инвестиций – никакой прибыли, а инвестиции могли быть значительными. Сверлить дырки в матушке земле – занятие не из дешевых.

³ Роско начинал работу на нефтяных полях. Позднее он перебрался в шахту. Играл он в гольф, он бы и там рыл землю клюшкой.

⁴ Трубой (tubular) называют 30-футовую бурильную трубу цилиндрического вида. Этим словом называют все, что имеет данный форм-фактор независимо от прочих характеристик. Трубы бывают очень тяжелыми, и их транспортировка никогда не бывает простым делом.

Звоню своему местному водителю,¹ и он присылает мне грузовую платформу. Через час все, что мне нужно, перевезено.

– Прелесть в том, – продолжал Роско, – что вся сделка заключается с помощью рукопожатия. Мы договариваемся о том, куда и когда надо перевезти трубы и сколько это будет стоить. Он берет на себя обязательство быстро доставить товар, а я – обязательство быстро ему заплатить. Контракт мы не подписываем, хотя я, возможно, чиркну свои инициалы на какой-нибудь бумажке, когда он будет забирать груз. Главное, нет никаких законников. Нет ни нужды в них, ни лишнего времени. Достаточно рукопожатия. Если один из нас не выполнит своего обязательства, мы никогда больше не станем работать вместе.

Вот как? Интересная мысль.

– Это честно. Твое слово – твое обязательство, а твое рукопожатие – это твое слово. И если пройдет слух, что твое рукопожатие ничего не стоит, ты оказываешься вне закона. Никто не станет иметь с тобой дела.

Как это связано с программным обеспечением

Так, подумал я, значит, Роско, рассчитывал на такую же железную верность своему слову от своих программистов. Можно допустить. Что же не сработало?

– Главная проблема заключается в следующем, – постулировал Роско. – На нефтяном поле обязательство означает «обязательство доставить». Помоему, эти ребята-программисты понимают обязательство иначе – как «обязательство постараться».

Вот оно! Роско без лишнего красноречия попал в самую точку. Я часто был свидетелем того типа провалов, который он описывал. Люди, только что сорвавшие срок выпуска продукта, начинают долго объяснять, какие большие усилия они приложили, как будто это снимет с них ответственность. Им кажется, что их старания служат доказательством благородных намерений, а потому их следует простить.

– Обязательство состоит из двух частей, – продолжал Роско. – Первая часть – это добровольное желание. Оно означает, что человек постарается сделать работу. Без желания ничего не будет. Но и вторая часть не менее важна. Это способность выполнить работу. Человек должен не только

¹ Наемный транспортировщик грузов. Эти ребята сидят у телефона и ждут таких заказов.

хотеть выполнить работу, но и *быть в состоянии* это сделать. Первое без второго бесполезно с точки зрения конечного результата.

– Поэтому, когда кто-нибудь берет передо мной некое обязательство, – заключил Роско, – я предполагаю, что он одновременно *хочет* и *может* его выполнить. Поэтому работа *будет сделана*. Дело закрыто. Если он не уверен в своих возможностях, не надо брать на себя обязательств.

Мне пришлось задуматься над этим глубже, чем я делал это раньше.

Мое домашнее задание съела собака

– Конечно, – продолжал Роско, – всегда найдутся жулики, которые станут сочинять сказки про то, как «мое домашнее задание съела собака». Я таких не терплю и всегда удивляюсь, откуда они вообще взялись. И знаешь что: они всегда сообщают тебе, что не успевают сделать работу вечером накануне сдачи, хотя ясно, что они знали об этом намного раньше. Это просто несерьезные люди, и их надо гнать как можно скорее.

Я решил, что это справедливо. Меня всегда возмущали те, кто ждал последней минуты, чтобы сообщить мне плохие вести. Я не только считаю это признаком их трусости, но и полагаю, что они украли у меня время, в течение которого я мог бы придумать альтернативный план, чтобы закрыть их провал.

– Ты правильно рассуждаешь, Роско, – отвечал я. – Но иногда кажется, что просто возникло какое-то недоразумение.

– Да, есть два распространенных типа «механического» провала. Во-первых, может быть разночтение по поводу того, что представляет собой обязательство. То есть что именно должен представлять собой результат? Сколько труб надо перевезти и в какое место в Техасе? Во-вторых, может быть нечетко определено «когда». Я всегда указываю водителю час, к которому трубы должны быть доставлены на место. Он понимает это так, что если он задержится с доставкой, то я не получу той услуги, за которую заплатил.

– Тут вопрос денег, – вставил я. – Применимо ли это, если речь идет о программном обеспечении?

Роско ответил без колебаний. – Программистам не мешало бы лучше уточнять, что именно должно быть поставлено и к какому сроку. Тогда меньше стало бы нытья относительно того, сделали они то, что нужно, или нет. Условия должны быть однозначными, четкими и не обсуждаться до бесконечности задним числом.

– Потому что обязательство, – подчеркнул он, – это контракт, в котором нет места двусмысленностям. В обязательстве не должно быть условий, напечатанных мелким шрифтом. Там не должно быть лазеек, позволяющих отвертеться, если уговор не выполнен в срок.

Я был просто поражен тем, как простая логика техасских нефтедобытчиков нашла применение в программном бизнесе. Но у меня тут же возникли некоторые опасения.

Войны спецификаций?

– Подожди, Роско, – вмешался я. – не так просто осуществить твоё намерение точно затвердить, что именно должно быть поставлено. Уокер Ройс неоднократно говорил мне, что стремление к излишней точности в самом начале проекта ведет к большим потерям времени. Например, если настаивать на очень точных условиях поставки, возникает масса споров относительно спецификаций.

– На самом деле, – отвечал Роско, – я согласен с Уокером. Трата уймы времени на обсуждение мелких деталей конечного продукта напоминает споры юристов о расстановке запятых в контракте. Желательно избежать подобных затрат. Иначе ничего никогда не будет доведено до конца, и мы потратим слишком много времени, плодя бесполезные документы. Но есть *реальная* проблема, которая требует решения. Вот пример. Очень часто инженер или программист обязуется написать за неделю к пятнице некий фрагмент кода, который должен участвовать в сборке более крупного проекта. В указанный срок код поступает, но в нем обнаруживаются ошибки. Еще неделя уходит на то, чтобы их устранить. Затем обнаруживается, что если число элементов в используемом им массиве возрастает со 100 до 1000, то алгоритм работает в 100 раз медленнее. На исправление этого уходит еще несколько недель. И все это время задачи, которые зависят от этого кода, висят и ждут, когда он будет доработан.

Роско разминался перед выступлением. Он продолжал.

– Проблема в том, что ребята просто не договорились о том, что именно должно было быть готово в ту пятницу. Программист считал, что это должен быть фрагмент кода. Менеджеру нужен был полностью отлаженный код, эффективно работающий в разумном диапазоне входных данных. Очевидно, что характеристики поставляемого продукта не были согласованы. Программист будет доказывать, что, предоставив код, он выполнил свое обязательство, и станет изворачиваться, объясняя, что он во-

все не обещал при этом выявить и исправить все ошибки. Но вступать на эту скользкую дорожку очень опасно, потому что теперь вы станете обсуждать, насколько серьезной может быть ошибка, которой разрешается проскочить тестирование (да проводил ли он его вообще?). А программист станет также доказывать, что вопросы эффективности вообще не обсуждались, хотя применение им алгоритма с квадратичной зависимостью от объема данных¹ вообще несовместимо с понятиями компетентности и профессионализма.

Интересно, кто это просветил Роско о квадратичных алгоритмах?

– Я не стану утверждать, что для решения этой проблемы нужны детальные спецификации, вплоть до точных показателей эффективности. Но так же, как я на своем нефтяном поле в Техасе вправе рассчитывать, что мои трубы после доставки на новую буровую не окажутся завязанными в крендель, так и менеджер программного проекта вправе полагать, что представляемый ему код не окажется полусырым. Особенно, когда программисту известно, что его код участвует в сборке проекта и будет использоваться другими людьми.

«Вот что я пытаюсь объяснить», – сказал Роско, откидываясь в кресле.

Три самых распространенных отговорок

Я деликатно поинтересовался у Роско, есть ли у него какие-либо представления о том, почему люди регулярно не выполняют свои обязательства. Стоило ли сомневаться, что у него было свое мнение по этому предмету?

– Сдается мне, что в программном бизнесе большинство обязательств ничего не стоят уже в тот момент, когда произносятся. Люди просто не дают себе труда хорошенько подумать, прежде чем принимать на себя обязательства. Если бы они понимали все последствия, вытекающие из принятия на себя обязательств, они были бы гораздо осторожнее. Приведу не-

¹ Входной массив вырос с 100 элементов до 1000, т. е. в 10 раз. Время обработки возросло в 100 раз, или 10 в квадрате. В такой ситуации мы говорим, что алгоритм имеет квадратичную сложность. Квадратичные алгоритмы легко программировать, но для всякой серьезной программы они убийственны. Хотелось бы, чтобы алгоритмы линейно зависели от размера входных данных. Знающий программист обнаруживает квадратичные (или еще менее эффективные алгоритмы) и заменяет их чаще всего такими, сложность которых не хуже $n \log(n)$. Иногда это лучшее, чего можно добиться.

сколько примеров. Вот одна из трех самых распространенных отговорок: «меня все время отвлекали, а я на это не рассчитывал». Позвольте, чьи это трудности? Явно не мои. Не я брал на себя обязательство. Тот, кто взял на себя обязательство, должен был либо, когда это делал, учесть, что его будут отвлекать, либо не отвлекаться, когда его пытались отвлечь. Очевидно, что обязательству, принятому в отношении меня, он не придал достаточного значения, если отвлекся на другие задачи. А как же понятие «ранее данного обещания»?

– Подожди, Роско, – воскликнул я, – всякое случается в жизни. Ты роешь ямы под столбы для забора, и начинается дождь. Ты вынужден прекратить работу. Разве не так?

– Да, иногда бывает дождь, – сказал он. – Я считаю, что, делая свою оценку, ты должен это учесть. Если 50% времени идут дожди, то не стоит в своей оценке полагаться на то, что дождя не будет. Нельзя делать такое предположение, а потом жаловаться, что пошел дождь.

– Да, – подумалось мне, – кажется, людям редко приходят в голову такие вещи. Многие обязательства исходят из того, что все будет идеально. Нежелание признать, что обстоятельства могут сложиться неудачно – одна из причин, приводящих к беде и невыполнению обязательств.

– Хорошо, что идет вторым в списке? – спросил я.

– Вот вторая отговорка: «задача оказалась сложнее, чем я предполагал». Опять-таки, чья это вина? Не я оценивал задачу, а он. Если у него не было уверенности, не надо было брать на себя обязательство. Если он недостаточно разобрался в задаче, чтобы правильно ее оценить, не нужно было брать на себя обязательство. Неожиданное прозрение посреди работы – явный непрофессионализм. То, что ты паршиво сделал оценку, не снимает с тебя ответственности за результат. Желание-то у тебя было, а компетентности не хватило.

Ох, сколько раз мне приходилось это слышать! Несчастный малый рассказывает, что трудился день и ночь, но задача оказалась намного, неизмеримо труднее, чем он предполагал. И он рассчитывает на сочувствие. Я часто обнаруживал, что мне жалко того человека, который только что меня подвел. Может быть, зря?

– Полагаю, – сказал я, – что иногда люди принимают на себя обязательства, потому что им кажется, что они сумеют сделать эту работу. И слишком часто, как мне кажется, они начинают что-то понимать, только вникнув в задачу глубже. Наверно, прежде чем брать на себя обязательство, они должны были попросить какое-то время, чтобы изучить задачу.

Роско кивнул. Очевидно, что он предпочел бы подождать более точной оценки, чем получить ненадежное обязательство. – Ну, – спросил я, – какая же третья стандартная отговорка?

– Последняя отговорка, – продолжал Роско, – это старая песня про то, как «меня подвел мой субподрядчик». То есть тот, кто взял на себя обязательство, передал часть работы кому-то третьему, а тот не сделал работу в срок. Что, это моя проблема? Да нет, черт возьми! Я никаких подрядчиков не нанимал, я даже не знал об их существовании. Вполне может быть, что этот третий не выполнил своего обязательства, но это не моя забота. Я договаривался с первым человеком. Ответственность передо мной несет он. Конец разговора.

Должен сказать, что и эту историю я уже слышал.

И еще одна...

Так значит, – сказал я Роско, – вся катастрофа свелась к тому, что кто-то не выполнил своих обязательств? Не слишком ли все просто?

– Можешь рассматривать это с каких угодно точек зрения, но мой ответ всегда будет один, – сказал Роско. – Да, есть люди, которые рисуют все эти диаграммы PERT и Ганта, пытаются спланировать все сверху донизу. Можно соглашаться с тем, что они делают, или не соглашаться, и разные умные люди имеют разное мнение по этому вопросу. Одно мне ясно наверняка: учесть все зависимости трудно, и всегда по ходу дела будут возникать новые задачи, которые не были учтены в плане. Все эти диаграммы напоминают мне паутину. Если отвлечься от деталей, то можно сделать вывод, что результат зависит от целостности паутины. Если хочешь, можешь называть ее сетью. Для меня это сложный набор звеньев в очень сложной n-мерной цепи.

Я пытался понять, куда клонит Роско. Едва ли он собирался читать мне лекцию по теории графов. Но я не стал вмешиваться и ждал.

– Далее я могу попытаться определить, что случится с моим графиком работы, если одно из звеньев разрушится. Если это звено входит в критический маршрут, я могу быть уверен, что весь проект будет задержан. Если маршрут не критический, общей задержки может и не произойти – смотря по тому, насколько задержится именно та конкретная область. Но все мы знаем, что проекты обычно опаздывают не из-за какого-то одного катастрофического провала – серьезного повреждения одного звена, – а из-за того, что есть много-много звеньев, в каждом из которых есть маленькая задержка, или, лучше сказать, каждое чуть-чуть надломлено. Поэтому, как я по-

нимаю, целостность всего графика – это суммарная целостность всех и каждого из его звеньев.

«А что есть каждое звено, – торжествующе закончил Роско, – как не взятое на себя обязательство? Моя гипотеза: в большинстве проектов неприятности происходят из-за накопления всех невыполненных обязательств. И это одна из причин, по которым менеджеры проектов обычно не могут понять, чем вызван провал. Не было какой-то одной большой поломки: просто накопилась масса мелочей. Это очень коварно».

Удар, защита, ответный удар

Во всем этом разговоре меня беспокоила одна проблема, и я решил, не откладывая, выложить ее перед Роско.

– Роско, – начал я, – мне понятно твоё желание привнести в процесс больше реализма и честности, но, честно говоря, мне кажется, что ты заблуждаешься. Разве можно сравнить простую детерминированную задачу с четкой целью, такую как перевозка труб в определенную точку, с выпуском большого и сложного программного пакета, когда неизбежны исследования, применение новых технологий и неопределенность? Ведь сравнивать такие задачи несправедливо!

– Извини, сынок, – ответил Роско, – но ты немного сбился. Правильной аналогией будет сравнить выпуск сложного программного продукта с получением первой бочки нефти из скважины. Кто тебе сказал, что это простая или детерминированная задача? Будь это так, мы никогда не сталкивались бы с сухими дырками. Но скажу тебе – я ни минуты не сомневаюсь в этом, – что вовремя поставить сколько-нибудь нефти можно только совместным выполнением всех взаимозависимых подзадач, включая своевременный подвоз труб. Я утверждаю, что все сложные проекты от бурения нефтяных скважин до разработки программ можно фактически разложить на мелкие конечные задачи. Каждую из них можно оценить с большей или меньшей степенью неопределенности. Но, в конце концов, каждая из этих задач и ее оценка представляют собой обязательство. Можно взглянуть на это иначе. Мы всегда стараемся работать в условиях, которые мы называем *атмосферой высокого доверия*. Это такая обстановка, в которой можно неявно положиться на товарища по работе. Если хочешь, «единицей измерения доверия» можно считать обязательство.

– Почему же тогда, – возразил я, – те, кто работает над большими проектами, так небрежно относятся к своим оценкам и обязательствам?

Жертва большого проекта

– Ну, это просто, – отвечал Роско. – Ребята из NASA с этим разобрались. Это называется *жертва большого проекта*. Во всех крупных проектах люди считают, что кто-то другой опоздает еще больше, чем они сами. Им кажется, что если они слегка задержатся, это будет незаметно на фоне чьей-то крупной задержки. Шутники в NASA утверждают, что когда идет обратный счет перед пуском, только в последние 10 секунд кто-нибудь не выдерживает и прерывает запуск. Все ждут, чтобы кто-то другой вышел из игры первым.

Это показалось мне серьезной проблемой. Роско подтвердил, что так оно и было.

– Да, тут извращенная психология, – сказал он в завершение. – Когда ты связываешь свой «успех» с провалом другого человека, то встаешь на гибельный путь. Но в этом и есть суть «жертвы большого проекта». Ты считаешь, что кто-то другой напортачит больше тебя. Это сильно подрывает мораль проекта и идею атмосферы высокого доверия.

Роско был снова прав, но я выпустил еще не все свои стрелы.

Конец программных разработок в прежнем понимании?

– Хорошо, Роско, – сказал я, – но твой подход порождает другую проблему. Если все проникнутся такой ответственностью за свои обязательства, как тебе хочется, не станут ли их оценки настолько консервативными, что, сложив их все в один график, ты обнаружишь, что не стоит и начинать проект, потому что теперь для него потребуется в три раза больше времени?

– Экая у тебя длинная получилась фраза, – подмигнул мне Роско. – Да, такое препятствие есть, и я знаю два обходных маневра. Во-первых, надо справиться с перестраховкой.¹ Ты должен добиться уменьшения таких консервативных оценок, при которых все задачи или обязательства окажутся выполненными. Надо добиться от людей таких оценок, чтобы их обяза-

¹ Sandbagging – перестраховка, такое завышение оценки времени, которое позволит без труда выполнить работу. Коммивояжеры имели обыкновение занижать оценку покупательной способности на своей территории, чтобы получить квоту продаж, которую им было бы легко превзойти. Их менеджеры могут иногда просмотреть это жульничество, но два года подряд такое случиться не может. Во второй раз торгового агента выгонят (или повысят до менеджера, как это уже случалось).

тельства оказывались выполненными в 80–90% случаев. Сегодня мы не достигаем и до 50%, что совершенно недопустимо. Во-вторых, стоит подумать о том, чтобы чаще закрывать проекты на ранних стадиях. Я слишком часто сталкиваюсь с графиками, которые лишь суммируют самые оптимистичные оценки для подзадач. Это не графики работы, а бесплодные мечтания. Такие проекты заранее обречены на провал. Исполнители говорят своим менеджерам то, что тем хотелось бы слышать, а менеджеры слушают, как завороченные. Проходит несколько месяцев или лет, результаты этого фарса обрушиваются вам на голову, и начинается охота на ведьм. ЭТУ ПОРОЧНУЮ ПРАКТИКУ НАДО ПРЕКРАТИТЬ!!!

На правом виске Роско вздулась вена, поэтому я предпочел немного отступить и смягчить свой заключительный выпад.

Проработка и построение системы

– А что ты скажешь по такому поводу, – начал я. – В Rational Unified Process средние две фазы, проработка и построение системы, занимают больше всего времени.¹ Я могу себе представить графики работы на основе обязательств для фазы построения системы, когда мы уже достаточно хорошо представляем себе, чем мы занимаемся. Но для этапа проработки, где все еще в большом объеме присутствуют исследования и риск, это проблематично.

– Хорошее замечание, – признал Роско. – Начнем с построения системы. Даже там мы сегодня много теряем из-за нарушения обязательств, поэтому усиление ответственности несомненно должно улучшить нашу эффективность. Взято на заметку. Кстати, я подозреваю, что обязательства так часто не выполняются на этапе построения системы потому, что на этапе исследования оценка трудоемкости задач была проведена плохо. Когда во время построения в массовом порядке нарушаются сроки, это служит верным признаком того, что предыдущий этап, этап проработки, был завершен преждевременно – возможно, из-за того, что поджимали сроки. Расплачиваться, разумеется, приходится на следующей стадии.

– Однако, – продолжил он, – вернемся к исследованиям. Нам еще предстоит научиться лучше определять объем и сроки исследовательских работ, которые, напомню тебе, можно разбить на более мелкие подзадачи

¹ Напомню все четыре фазы: исследование, проработка, построение системы и передача в эксплуатацию.

с меньшей степенью неопределенности и с лучшими возможностями для оценки сроков. Нам нужны обязательства с жесткими сроками и на стадии проработки, чтобы стимулировать принятие решений. В противном случае мы будем вечно изучать проблему. Полезно устанавливать срок для принятия решений по рискованным вопросам. Это концентрирует усилия и гарантирует подведение окончательной черты.

Мне больше нечего было возразить. Как всегда, Роско продумывал партию не на два хода вперед, а на шесть.

Жестокая любовь

– Итак, – сказал Роско, – спускаемся на землю. Я сажусь со своими менеджерами и учу их жизни. Отныне мы делаем честные оценки и даем реальные обещания. И пусть пеняет на себя тот, кто еще раз позволит себе не сделать работу вовремя.

– Кстати, – сказал он застенчиво, – я и сам больше не засну за рулем.

Мне показалось в тот момент, что первый промах Роско будет у него и последним.

Резюме

Материал этой главы вызвал более горячие дебаты, чем вся остальная книга. Разработчикам программ и их менеджерам очень не нравится мое жесткое отношение к принятым обязательствам. Они все время пытаются мне доказать, что «разработка программ – это очень специфическая область».

Извините, ребята, но я с вами не согласен. В любых сферах человеческой деятельности существуют риск и неопределенность. В каждом из когда-либо предпринятых проектов был элемент новизны. У всех проектов слишком сжатые сроки и недостаточные ресурсы. Поинтересуйтесь у своих друзей, работающих в других областях. Не настолько вы исключительны.

Дело не в том, что я не способен посочувствовать. В разработке программ есть свои особые проблемы. Одна из наиболее зловредных среди них – заблуждение руководства по поводу того, что в последнюю минуту можно все переиначить, стоит лишь чудесным образом заменить одну строку кода другой. Так не бывает, это вымысел. Скорее всего, при выявлении проблемы потребуются существенная переработка продукта, как и во всякой другой области, где результат определяется архитектурой.

Часто можно услышать другой аргумент: дескать, в любом грандиозном программном проекте есть новые и уникальные особенности, затрудняющие планирование и своевременную поставку и в принципе превращающие его в научно-исследовательскую задачу. У меня есть два возражения:

- Во-первых, если построить процесс так, чтобы исследовательская часть проекта была ограничена и замкнута в фазе «проработки» (как в Rational Unified Process), то ее можно контролировать и удерживать в рамках. Здесь может сильно помочь итеративная разработка, нацеленная на уменьшение рисков.
- Во-вторых, я готов держать пари, что большинство программных проектов не укладывается в сроки не из-за затягивания исследовательской стадии, на которую обычно выделяется относительно небольшое время. Нет, чаще проект опаздывает из-за плохого выполнения рутинной части работы, а исследования и разработки становятся козлом отпущения. Легче заслужить прощение за срыв той части проекта, которая связана с высокой наукой, чем отпущение грехов за неспособность справиться с самыми обыденными вопросами.

С чем бы я хотел раз и навсегда покончить, так это с представлением о том, что ПО обладает какими-то особыми свойствами, из-за которых нельзя требовать от разработчиков, чтобы они отвечали за взятые на себя обязательства. Эта идея неправильна и опасна. Она всегда служит оправданием бездеятельности и в значительной мере обуславливает сегодняшний непрофессионализм в разработке ПО. Профессионалы серьезно относятся к своим обещаниям и всегда отвечают за них. Это важнейшее понятие как для отдельных лиц, занимающихся разработкой программ, так и для профессии в целом.

В связи с этим есть важное указание и для менеджеров: не вынуждайте работников брать на себя обязательства, которые они считают нереалистичными, но которые устраивают вас по срокам. Вы можете вынудить их к этому своим давлением, но эта победа будет слабым утешением, когда окажется, что обязательства не выполнены. Только тогда у обязательств есть достаточные шансы быть выполненными в срок, когда они достигнуты в результате компромисса между менеджером и разработчиком, который оставил каждого из них с чувством легкой неудовлетворенности.

В следующей главе я перехожу к теме, которую открыто никогда не обсуждают. Это деликатная тема вознаграждения за труд. Почему так мало советов можно найти в этой области? Вы найдете некоторые новые интересные идеи по этому поводу на следующих страницах.

ГЛАВА ШЕСТНАДЦАТАЯ

Вознаграждение

Вопросы вознаграждения профессиональных программистов всегда вызывают споры. Лучший разработчик в команде часто производительнее худшего раз в десять, и никакие обычные системы зарплаты это учесть не могут. В этой главе я рассмотрю некоторые новые взгляды на денежное вознаграждение и продуктивность в организациях, занимающихся разработкой ПО. Я не изобрел «универсального средства» решения этой проблемы, но предлагаю нетрадиционный взгляд на нее.

Важный вывод: оказывается, что многие задачи, которые мы пытаемся решить с помощью денежной компенсации, обусловлены проблемами распределения заданий и повышения квалификации и требуют применения средств, разработанных для соответствующих областей.¹ Я пытаюсь увязать три переменных: вознаграждение,² квалификацию и сложность задачи. Вознаграждение связано с эффективностью труда, поэтому надо ус-

¹ Заявление: излагаемые здесь идеи принадлежат автору книги. Они не выражают философии, политики или практики, существующих в Rational Software. Автор многое почерпнул из пространных и глубоких бесед со своим сыном Марком, который является семейным специалистом по денежным вознаграждениям.

² Я говорю только об одной стороне вознаграждения – финансовой (зарплата, премии, опционы на акции и т. д.). Я *не* говорю о завуалированном вознаграждении (большой угловой кабинет с окном, известность, власть, влияние). Однако хочу подчеркнуть, что вознаграждение не ограничивается одним лишь денежным довольствием: в распоряжении руководства есть многочисленные механизмы вознаграждения, и в соответствии с особенностями конкретного работника и профилем организации может быть применен самый подходящий из них.

становить связь между эффективностью с одной стороны и уровнем мастерства и сложностью задачи с другой.

Ищем поток

Рассмотрим сначала работу Михая Чиксентмихайи (Mihaly Csikszentmihalyi),¹ в которой утверждается наличие у человека *канала потока*, что примерно можно перевести как напряженную сосредоточенность и чувство удовлетворения, испытываемые, когда уровень нашего мастерства соответствует задачам, которые мы решаем в нашей работе.

Если задача слишком простая, становится скучно, и человек не чувствует себя счастливым (рис. 16.1). Если она оказывается очень сложной для уровня подготовки человека, он испытывает напряжение и тревогу. Если возникает умеренное соответствие – не слишком просто, но и не слишком сложно, – то мы попадаем в поток, который охватывает достаточно широкий диапазон уровня компетентности и сложности задач. В этом благодатном состоянии мы многого добиваемся и испытываем чувство удивительного благополучия; спортсмены описывают его как «нахождение в зоне».²

Обратите внимание, что из позиции 1 на рис. 16.1 человек может попасть в зону тревоги, если сложность задачи увеличится, его мастерство не вырастет соответствующим образом (позиция 2). Чтобы вернуться в поток, достаточно любой комбинации роста мастерства и снижения трудности, но лучше всего двигаться по горизонтали в позицию 3. Аналогично можно повысить мастерство при неизменной сложности работы и переместиться из позиции 1 в позицию 2', где наступает скука. Для входа в поток тогда потребуется двигаться в сторону позиции 3, т. е. искать более трудную работу. Конечно, позиция 3 станет тогда новой позицией 1, и весь цикл повторится. Однако в позиции 3 человек приносит больше пользы,

¹ Mihaly Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. (New York: HarperCollins, 1991) Чиксентмихайи М. «Поток: психология оптимального переживания» (Смысл, 2006)

² В области разработки программного обеспечения человеком, кратко выразившим это состояние, стал легендарный Говард Ларсен (Howard Larsen), один из создателей Rational Software. В этот момент каждый, кто работал с Говардом, согласно кивнет головой: Говард, наверно, входил в поток настолько глубоко, насколько это возможно для программиста. Уверен, что в вашей организации либо есть такой Говард, либо вам хотелось бы его иметь. По-настоящему следует задаться вопросом: почему Говарды столь редки? Говардами рождаются или становятся, и если становятся, то как создать их побольше?

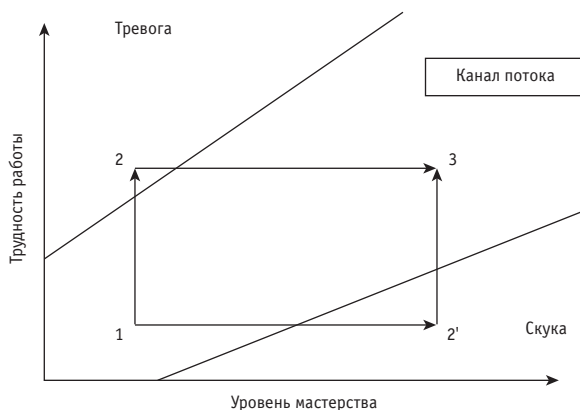


Рис. 16.1. Модель потока

чем в позиции 1, хотя оба они соответствуют состоянию потока. Очевидно, что перемещение вверх и вправо приносит хороший результат как члену команды, так и организации. Я полагаю, что канал потока «открывается» при росте уровня мастерства и сложности работы, поэтому теоретически у людей появляется больше возможностей получить удовлетворение от работы.¹

Следует также отметить, что нахождение в потоке ведет к *оптимальной* продуктивности. Даже находясь вне потока, можно достичь такого уровня продуктивности, который достаточен для оценки «удовлетворительно» во многих организациях. Однако чем больше человек отдаляется от состояния потока, тем ниже его производительность. Возникает законный вопрос: *а почему не поставить задачу вывести каждого сотрудника организации на оптимальный уровень продуктивности?* Разве мы не получим в результате максимальный выигрыш как для индивида, так и для организации?

В действительности содержание работы Чиксентмихайи гораздо богаче, чем можно показать в таком кратком описании. Однако и этого достаточно для принципиального понимания концептуальной модели, от которой я отталкиваюсь, говоря о вознаграждении.

¹ Кажется, теория Чиксентмихайи не содержит такого положения, но мне оно представляется разумным дополнением ее. Именно такое «открытие» канала приводит к появлению трехмерного конуса, который я продемонстрирую далее в этой главе.

Поток и продуктивность разработки ПО

Прежде чем углубиться в вопросы вознаграждения, посмотрим, как понятие потока связано с продуктивностью/производительностью в сфере разработки ПО. Одно из самых интригующих и загадочных явлений, с которыми сталкиваются менеджеры программных разработок, – это огромное различие в продуктивности отдельных участников команды. Чем можно его объяснить?

Мой опыт работы в большом количестве проектов и с еще большим количеством членов команд показывает, что продуктивность архитекторов, конструкторов, программистов и тестеров, которые нашли для себя состояние потока, значительно превосходит результаты их коллег, которым это не удалось.¹

Для тех, кто показывает низкие результаты производительности, характерны две черты, обозначенные Чиксентмихайи: тревога от своей беспомощности (сложность задачи намного превосходит уровень мастерства) или скука от легкости проблем (уровень мастерства намного превосходит сложность задачи). Результат? Плохой код из-за неопытности и необходимости изобретать колесо в первом случае или из-за промедления и дерзости («да если будет надо, я склепаю это за минуту!») во втором.

Конечно, другой важный фактор продуктивности обусловлен культурой компании, которая может «платить» за результат или за отображение работы или человека на бумаге. Здесь существует некоторая путаница в понятиях. По моему разумению, мастерство представляет собой потенциал для выполнения работы. Благодаря тренировке и опыту мы приобретаем мастерство, которое позволяет нам достичь высокой продуктивности, но не гарантирует ее.

Аналогично трудность задачи дает *возможность* продемонстрировать высокую продуктивность. То есть особенно сложная задача создает ситуацию, в которой сочетание усилий и мастерства может дать замечательные результаты. У человека с высокой мотивацией и мастерством отсутствие целей вызывает разочарование, поскольку у него нет возможности проявить себя. Но, как и с мастерством, одного лишь наличия сложной задачи недостаточно для демонстрации высокой продуктивности.

¹ В оценке продуктивности я учитываю как количество, так и качество созданных программных объектов.

В идеале мы хотим платить за продуктивность, т. е. за *результат*, или суммарный вклад в работу организации. А это означает, что надо разобаться во *всех* факторах, влияющих на продуктивность. В их число входят не только соответствие или несоответствие между сложностью задачи и мастерством. Например, мощным фактором является мотивация. Кроме того, люди гораздо эффективнее работают, когда заняты тем, что им нравится. Еще один фактор – уверенность в своих силах: высокая продуктивность может создавать положительную обратную связь, которая еще больше усиливает уверенность в себе и увеличивает продуктивность.

Когда человек находится в состоянии потока, мотивация и удовольствие от работы возрастают максимально, и уверенность в себе стремительно растет. Поэтому концепция потока согласуется с другими факторами, влияющими на продуктивность/производительность.

Однако не следует пренебрегать факторами мотивации, которые не зависят от потока. В частности, нельзя недооценивать мотивационный эффект приверженности задаче и ощущения себя частью отличной команды. И, конечно, циклы повышенной и пониженной мотивации сменяют друг друга в зависимости от событий в личной жизни людей.

Наконец, как мотивирующим, так и демотивирующим фактором может служить *восприятие* вознаграждения. Те, кто считают, что получают неадекватное вознаграждение – в абсолютном значении или относительно коллег – могут утратить мотивацию настолько, что пострадает их продуктивность. (Проблема здесь, конечно, в том, что несправедливость может существовать только в личном *восприятии*: работник может переоценить свой вклад и/или вознаграждение, полученное коллегами). Поэтому здесь возникает еще одна обратная связь, которую надо иметь в виду.

Хотя все эти факторы влияют на продуктивность, а следовательно, и на вознаграждение, в оставшейся части главы я буду придерживаться модели потока Чиксентмихайи, хотя она чрезмерно упрощает очень сложную ситуацию. Я принял такое решение, потому считаю возможным непосредственно оказывать воздействие на три фактора – уровень мастерства, сложность работы и размер вознаграждения, тогда как управлять остальными упомянутыми мной факторами гораздо сложнее.

Применение модели потока к вознаграждению

Далее я предполагаю наличие трех зон вознаграждения: адекватного вознаграждения, с недоплатой, с переплатой. *Переплата* означает,

что вознаграждение данного участника команды превышает его вклад в работу организации. *Недоплата* означает, что вознаграждение данного участника команды ниже его вклада в работу организации.¹ Об *адекватном вознаграждении* можно говорить, если вклад данного участника команды пропорционален его вознаграждению для *данной* работы и *данной* организации; относительно других контекстов никаких предположений не делается.² И ясно, что тот, кто плохо делает свое дело, всегда по определению находится в зоне переплаты.

Модель, основанная на мастерстве

В некоторых организациях вознаграждение увязывается с уровнем мастерства, при этом сложность работы не считается доминирующим фактором. Такая система шире распространена в правительственных организациях и там, где учитывается возраст – т. е. считается, что со временем мастерство возрастает. В рамках модели «потока» этот подход отражен на рис. 16.2.

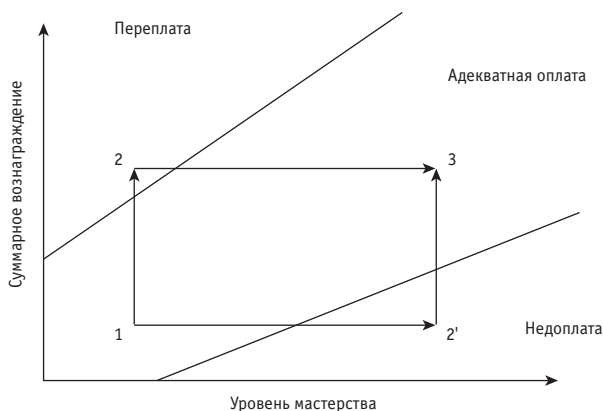


Рис. 16.2. Модель вознаграждения, основанного на сложности задачи

¹ Предполагается, что вклад в работу организации может быть объективно и сообразно измерен.

² Следует упомянуть досадную проблему «наград за редкость», которую выплачивают обладателям некоторых видов мастерства в условиях временного несоответствия рыночного спроса и предложения на них. В данный момент я не учитываю это явление, но вернусь к нему позднее в этой главе.

Как видите, есть зона «адекватного вознаграждения», в которой уровень вознаграждения соответствует объему навыков участника команды, занятого данной работой. В этой зоне находятся позиции 1 и 3, хотя последней явно соответствует большее вознаграждение, чем первой.

В позиции 2 член команды зарабатывает больше, чем приличествовало бы его уровню мастерства. Такое положение может свидетельствовать о проблеме, но не обязательно. Например, работник может заниматься более сложной задачей, требующей большего мастерства, чем то, которым он владеет, поэтому он получает соответствующую компенсацию. А может быть, это просто ошибка.

В позиции 2' члену команды недоплачивают за его мастерство. Опять же для этого могут быть разные причины. Возможно, работник действительно не полностью загружен работой либо эта работа не требует того уровня мастерства, которым он владеет.

Модель, основанная на задаче

В некоторых организациях меньше интересуются уровнем мастерства и определяют вознаграждение в соответствии с трудностью задачи (либо вместо этого с мерой ответственности). Такая модель более типична для предпринимательской среды, начинающих компаний (рис. 16.3).¹

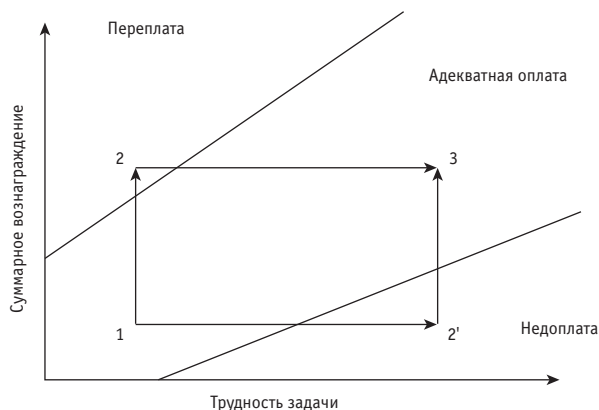


Рис. 16.3. Модель вознаграждения, основанная на уровне мастерства

¹ Это также относится к вознаграждению руководителей компаний. Оно часто определяется сравнением с аналогичными должностями в фирмах аналогичного размера.

Здесь позиции 1 и 3 также соответствуют правильному вознаграждению, и в последней вознаграждение выше, чем в первой.

В позиции 2 участник команды получает чрезмерное вознаграждение относительно сложности работы. Это может происходить по ряду причин. Может быть, по ошибке, а может быть, исполнитель этой работы обладает высоким мастерством, не востребованным в данной работе, или у него очень хорошие рекомендации.

В позиции 2' участнику команды недоплачивают. Так бывает, когда человек выдвинут на новую и более сложную работу, но соответствующее вознаграждение ему еще не назначено.

Обратите внимание, что в этой модели, как и в предыдущей, попадание работника в положение 2 можно исправить, повысив его квалификацию или поручив более сложное задание при сохранении уровня вознаграждения. Если участники команды находятся в положении 2', то надо увеличить вознаграждение.

В любой модели вход в «зону» за счет снижения сложности задачи, уровня мастерства или величины вознаграждения всегда сопряжен с трудностями. Обычно в организациях не приветствуются изменения в сторону понижения. Ниже я остановлюсь на этом подробнее.

Конус адекватного вознаграждения

Теперь я объединю эти модели в одну трехмерную диаграмму и представлю вам «конус адекватного вознаграждения» (рис. 16.4).

Этот конус иллюстрирует состояния, в которых уровень мастерства, сложность задания и размер вознаграждения находятся во взаимном равновесии. Область внутри этого конуса, помеченная буквой «А» в центре проекции, соответствует классической ситуации «взаимного удовлетворения». Участник команды не испытывает ни скуки, ни тревоги. Он находится в состоянии потока. Вознаграждение также справедливо: участнику команды платят не больше, но и не меньше того, что он заслуживает. Он действует с максимально возможной для него продуктивностью, и организация честно расплачивается с ним за это. Обратите внимание на расширение конуса вверх: с ростом всех переменных появляется больше свободы для назначения правильного вознаграждения.¹

¹ Это справедливо и на практике: при низких значениях параметров легче допустить ошибку.

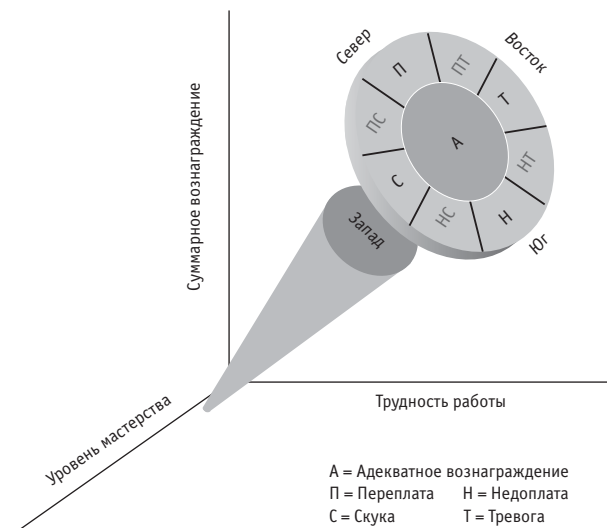


Рис. 16.4. Конус адекватного вознаграждения

Остальные восемь зон соответствуют возможным сценариям «непопадания в конус». Я обозначил их как стороны света на компасе, чтобы упростить дальнейшее обсуждение.

Принципиально возможно, хотя это может быть сопряжено с трудностями, выправить положение, перемещаясь в направлениях В и З. При этом участники команды будут получать адекватное вознаграждение, но испытывать скуку или тревогу. В этих областях продуктивность не достигает оптимальной величины, но поскольку проблемы связаны с несоответствием мастерства/сложности, манипуляции размером вознаграждения бесполезны.

С трудностями, расположенными по направлениям С и Ю, справиться относительно несложно. В данном случае участник команды находится в состоянии потока, но при этом его вознаграждение недостаточно или чрезмерно. Необходима корректировка для восстановления равновесия, так чтобы ни участник команды, ни организация не извлекали чрезмерной прибыли из его нахождения в состоянии потока.

В следующем разделе мы обсудим остальные четыре направления, представляющие собой несколько более патологические случаи. Еще один раздел будет посвящен детальному рассмотрению сценариев В, З и С, Ю, которые я только что описал. В обоих разделах я беру проекцию конуса

и рассматриваю ее как двумерный график. Четыре диагональных случая представлены в следующем разделе таблицей 2×2, а затем центральная область и восемь направлений компаса – в таблице 3×3.

Диагональные случаи

В центре внимания данного раздела четыре крайние ситуации, в которых члены команды не соответствуют выполняемой ими работе и получают неправильное вознаграждение. Этим ситуациям соответствуют четыре расположенных по диагоналям позиции на конусе – СВ, СЗ, ЮВ и ЮЗ соответственно (рис. 16.5).

Пожалуйста, обратите внимание, что «неадекватное вознаграждение» может иметь массу причин, включая ошибки в прошлом, временный дисбаланс на рынке труда между спросом и предложением работников определенной квалификации и т. д.

В левом верхнем квадрате (СЗ) находятся те члены команды, которые получают вознаграждение, соответствующее их квалификации, но выполняют относительно простую работу, которую в организации могли бы за меньшие деньги выполнять менее квалифицированные сотрудники. Это плохой квадрат: продуктивность по существу не соответствует уровню оплаты, и организация несет убытки упущенных возможностей, поскольку не полностью реализует потенциал своих работников.

Левый нижний квадрат (ЮЗ) – это те, кому скучно, да и платят им недостаточно. Некоторая квалификация у них имеется, работа относительно

	Скука	Тревога	
Переплата	Относительно простая работа, оплачиваемая по квалификации; убыток для организации	Оплата за труд, компетентность не растет (принцип Питера)	С
Недоплата	Недозагрузка	Тяжелая работа; перегрузка, оплата по квалификации; работников надолго не хватает	Ю
	З	В	

Рис. 16.5. Несоответствие участников команды своей работе или неадекватное вознаграждение

простая, а платят не много. Иногда сочетание неудовлетворенности работой и вознаграждением побуждает их искать другую работу; часто они просто прозябают. Этот класс представляет непроизводительную трату человеческого капитала. В эту категорию попадают некоторые государственные служащие.¹

В правом нижнем углу (ЮВ) располагаются самые угнетенные: они выполняют тяжелую работу, для которой их квалификации не вполне достаточно, и при этом им недоплачивают. Такое сочетание нежизнеспособно, поскольку рано или поздно эти люди окончательно измучаются.

Наконец, правый верхний (СВ) угол представляет тех участников команды, которым платят повышенное вознаграждение за то, что используют их сверх того, что предполагает их квалификация. Какое-то время они могут продержаться, но не долго, если не повысят свое мастерство. Поскольку их материальная заинтересованность в том, чтобы «прыгнуть выше головы», сильна, это прекрасные кандидаты для доказательства принципа Питера.²

Девять вариантов

На рис. 16.6 представлены все девять возможных случаев для нашей конической модели, а не только четыре самых патологических, показанных на рис. 16.5.

Здесь конус адекватного вознаграждения отображен «десяткой» в центре: достигнуто состояние потока, и нет претензий к вознаграждению.

¹ Не считайте это необоснованным выпадом в отношении государственных служащих, многие из которых напряженно трудятся и преданы своему делу. Это обвинение против тех видов работы, которые им часто предлагают. А поскольку они многочисленны, зарплата их оказывается невелика.

² *Принцип Питера* гласит, что в любой организации люди продвигаются по службе, пока не достигнут уровня своей некомпетентности. Одна из причин этого в том, что существует экономический интерес (большее вознаграждение), побуждающий взяться за более сложную или ответственную работу. Такие работники рассуждают, что если они «смогут удержаться», то лучше страдать и получать при этом большое жалованье, чем маленькое. Поскольку в большинстве организаций не принято понижать служащих в должности, существует тенденция продвигаться вверх до того момента, когда становится очевидной невозможность дальнейшего повышения. Обычно это приводит к тому, что человек останавливается в своей карьере на одну или более ступеней выше своего уровня компетенции. Этим феноменом объясняют впечатление широко распространенной некомпетентности руководства во многих организациях.

Участник команды доволен и продуктивен, а организация получает от него максимум производительности по справедливой цене.

В среднем ряду слева и справа (З и В) вознаграждение оказывается справедливым, но участник команды и организация несут потери, потому что продуктивность неоптимальная. В этом случае манипуляции с вознаграждением не решают проблем, из-за которых участник команды недоволен, а организация не полностью реализует потенциал работников. Это в чистом виде несоответствие сложности/мастерства, и решать проблему надо по этим двум направлениям.

В средней колонке сверху и снизу (С и Ю) участник команды находится в потоке, а продуктивность максимальна. Вознаграждение должно быть скорректировано таким образом, чтобы ни работник, ни организация не извлекали из этой ситуации сверхприбыли. Очевидно, что в случае недоплаты это сделать легко. Если же вознаграждение чрезмерно, обычно сохраняют его на том же уровне, пропорционально увеличивая при этом другие параметры (сложность работы и компетенцию).

Четыре случая по углам были рассмотрены в предыдущем разделе.

В следующем разделе я рассмотрю вопросы устойчивости этих состояний, а также действия, которые могут предпринять менеджеры для достижения обоюдной удовлетворенности участников команды и организации.

		Скука	В потоке	Тревога		
Плата	Переплата	Простая работа, оплачиваемая по квалификации; убыток для организации	Продуктивные участники команды, организации немного хуже	Оплата за труд, компетентность не растет (принцип Питера)	С	
	Адекватная оплата	Неоптимальная эффективность, в убытке все	Обоюдное удовлетворение	Неоптимальная эффективность, в убытке все		
	Недоплата	Недозагрузка	Продуктивные участники команды, организации немного лучше	Тяжелая работа; перегрузка, оплата по квалификации; работников надолго не хватает	Ю	
		З		В		

Рис. 16.6. Девять вариантов в конусе правильного вознаграждения и рядом с ним

Нестабильность состояний «север», «юг», «восток» и «запад»

Мы знаем, что центр на предыдущем рисунке соответствует оптимальному состоянию, и когда участник команды туда попадает, надо перемещать его вдоль «диагонали» конуса, чтобы он оставался в области правильного вознаграждения.

Мы знаем также, что четыре угла – это гиблые места, где положение быстро ухудшается или влечет постоянные убытки для участника команды и/или организации.

Устойчивы ли промежуточные состояния? Хотя они не столь опасны, как угловые, но с течением времени теряют устойчивость, если не предпринимать никаких действий (рис. 16.7).

В обоих случаях в среднем ряду (В и З) участник команды, получающий адекватную оплату и не находящийся в состоянии потока, становится все менее продуктивным по причине тревоги или скуки. В какой-то момент он переместится по диаграмме вверх, как показано стрелками, помеченными буквой «А», попав в результате в категорию с чрезмерным вознаграждением. Это произойдет без всякого номинального увеличения вознаграждения, а в результате снижения продуктивности при сохранении вознаграждения.

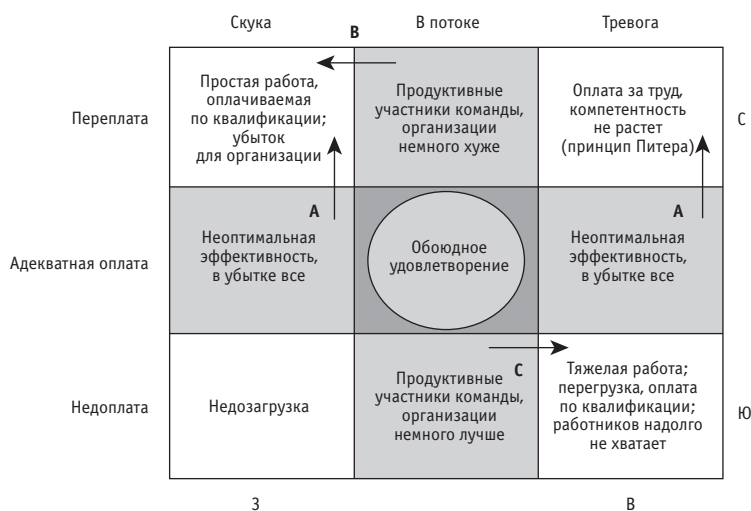


Рис. 16.7. В отсутствие управления средние состояния смещаются в вершины

Мы видели, что находящиеся в потоке члены команды, которым переплачивают (С), могут заскучать. В результате их продуктивность снизится, и они переместятся в угол СЗ, как показано стрелкой с меткой «В». Мне также случалось наблюдать, как находящиеся в потоке члены команды, которым недоплачивают (Ю), становятся тревожными, видя несправедливость получаемого ими вознаграждения. В результате возникает тенденция к выходу их из потока и перемещению в угол ЮВ, как показано стрелкой с меткой «С». (По мере снижения их продуктивности степень недоплаты за их труд уменьшается.)

Достижение обоюдного удовлетворения

Правильное руководство должно иметь своей целью перемещение всех участников команды в область обоюдного удовлетворения внутри конуса и удержание их там. Большинство организаций не обладает гибкостью, которая позволила бы им легко снизить сложность работы для своих сотрудников, так же как маловероятно, чтобы там захотели (или смогли) уменьшить денежное вознаграждение. Бессмысленно также обсуждать снижение квалификации как способ достижения равновесия,¹ хотя иногда такой эффект дает перемещение по горизонтали – иными словами, какие-то навыки работника могут не понадобиться в новой должности, зато потребуются приобрести новые. Но обычно единственный способ установить равновесие – это сдвиг каких-то параметров в положительном направлении, а не в отрицательном.

Большинство рекомендуемых средств, показанных на рис. 16.8, говорит само за себя, но на некоторых особых случаях мы остановимся отдельно.

В углу СЗ единственным лекарством может быть резкое увеличение сложности работы или ответственности. Участник команды обладает высокой квалификацией и уже получает вознаграждение, соответствующее более сложным задачам.

В квадрате С надо постараться пропорционально поднять уровень мастерства и сложность задачи, чтобы участник команды не вылетел из потока. А в квадрате СВ – в большей мере поднять квалификацию, чем сложность работы, иначе грозит катастрофа.

Повысить квалификацию не так просто. Обычно за день этого не сделать, и могут потребоваться обучение, тренировка, наставничество и другие способы, требующие времени. Работников, обладающих потенциалом,

¹ И во всяком случае постарайтесь обойтись без лоботомии.

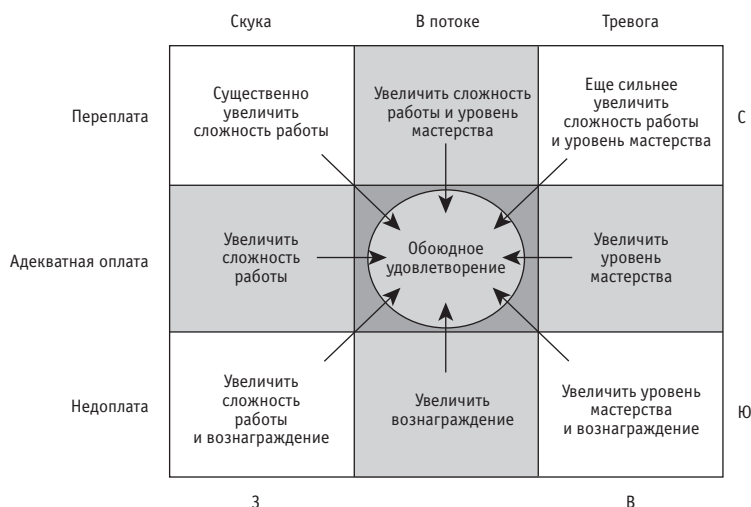


Рис. 16.8. Достижение обоюдного удовлетворения

но нуждающихся в повышении квалификации, следует выявлять *заранее*, чтобы принять корректирующие действия до того, как ситуация обострится настолько, что ее нельзя будет исправить в разумные сроки.

Обратите внимание, что из всех восьми неоптимальных ситуаций только одну можно корректировать простым увеличением вознаграждения. В пяти случаях не требуется изменять вознаграждение, но надо корректировать квалификацию и сложность работы, чтобы попасть обратно в конус. В оставшихся двух случаях необходимо изменить либо уровень квалификации, либо сложность работы в сочетании с вознаграждением. Примечательно, что изменение вознаграждения оказывается частично благотворной мерой лишь в трех из восьми проблемных ситуаций.

Распределение участников команды по классам

Если вы руководите командой, проделайте такой эксперимент. Нарисуйте таблицу размером 3×3 с клеткой «все довольны» в середине и распределите всех своих сотрудников по девяти ящикам. Вы увидите своих людей в новом свете, в каком еще не приходилось их видеть. Вы обнаружите, что в одну и ту же клетку попали люди, различные в прочих отношениях.

После этого у вас появятся мысли относительно плана развития и стратегии вознаграждения для каждого из них. Это упражнение заставляет взглянуть на оба эти аспекта одновременно, что мы иногда забываем сделать. Эта методология также позволяет более конструктивно беседовать с членами команды, потому что они увидят, что, рассматривая их продвижение, вы учли все переменные.

Распределение по классам кандидатов для найма

Подобрав себе работников на рынке труда с ограниченным предложением, нам иногда приходится платить им больше, чем того хотелось бы. Становится не так-то просто обеспечить справедливость по отношению к тем, кто работает уже давно. А иногда, чтобы переманить к себе работника, приходится предлагать ему условия, лучшие, чем те, которые он имеет в данный момент в другой организации, но предоставленные ему в силу какой-то ошибки.

В таких ситуациях можно предложить расписать свой нынешний личный состав по сетке и посмотреть, куда в ней попадет новый игрок. Если выяснится, что несправедливость устанавливаемой зарплаты породит трудности не только сейчас, но и в долгосрочной перспективе, то, возможно, стоит пересмотреть предлагаемые условия.

Деньги не всегда помогают

Эту главу можно было бы назвать: «Когда проблема вознаграждения не является проблемой вознаграждения?» Менеджеры, столкнувшиеся с трудностями, часто подчиняются импульсу и пытаются корректировать вознаграждение. Но очень часто трудности связаны с неправильным поведением, которое отчасти может быть вызвано размерами вознаграждения, а также и другими важными причинами. Простое изменение вознаграждения (что всегда означает его увеличение) не всегда приводит к желаемому результату. Метод анализа на конусе дает возможность рассмотреть все параметры одновременно. Если вам кажется, что участник команды не попадает в конус, первым делом надо выяснить, в которую из восьми ячеек он попадает. Только после этого можно искать решение.

Проводя такого рода анализ, мы также напоминаем себе о необходимости лучше разбирать связи между планами роста сотрудников и траекториями их заработков. Рассматривать одно в отрыве от другого хуже, чем решать проблему в два приема. Если не взглянуть на ситуацию в целом и не

предложить план, который одновременно учтет повышение квалификации, сложность задач и размер вознаграждения, то обе части уравнения могут резко дать себя знать. Метод конуса дает логичный способ изучить картину в целом и помогает более успешно сохранять и продвигать участников команды, а также поднимать их продуктивность.

Прочтя черновик этой главы, один очень опытный менеджер¹ заметил: «Я думаю, что это очень важное положение, но весьма опасаясь, что менеджеры не поймут, в какой мере оптимальность результатов зависит от правильного сочетания задачи, квалификации и мотивации. Когда распределение заданий основано на размере вознаграждения, это катастрофа. На самом деле размер вознаграждения не имеет отношения к правильному распределению задач. Оптимальное вознаграждение должно быть результатом оптимального распределения задач, а не наоборот».

Итак, если принять такой порядок приоритетов, то одним из ответов на коан² о правильном вознаграждении может быть конус адекватного вознаграждения. Эта технология поможет вам подобрать для сотрудников правильную работу с правильным уровнем квалификации и создать им мотивацию для оптимальной производительности. После этого проще определить для них правильное вознаграждение.

Резюме

Мне иногда задают такой логичный вопрос: «Из двумерного графика Чиксентмихайи, показывающего связь трудности задания и уровня мастерства, можно заключить, что участник команды более продуктивен внутри потока и менее продуктивен вне его. Можно ли на основе этой информации предсказать суммарный вклад в работу организации как функцию трудности задания и уровня мастерства?»

Отвечу, что можно попытаться смоделировать эту информацию. Такая модель будет очень полезна, поскольку если знать вклад, можно попробовать установить более тесную связь между ним и вознаграждением.

¹ Джон Ловитт (John Lovitt), долгие годы работавший в Rational Software.

² Concise Oxford Dictionary определяет коан так: «Загадка, с помощью которой в дзен-буддизме демонстрируется неадекватность логического рассуждения. [яп. = открытый план]». Надеюсь, мне удалось показать, что правильное вознаграждение может не быть загадкой, не поддающейся логическим рассуждениям.

Вспомним, что в нашей базовой модели канала потока (см. рис. 16.1) канал состоит из двух прямых, направленных вверх и вправо. Сделаем несколько допущений. В нашей упрощенной модели эти две прямые параллельны и канал потока не «открывается». Я предполагаю, что факторы, оказывающие влияние на продуктивность вне модели потока, несущественны в сравнении с факторами, действующими в модели потока. Я уже говорил, что это сильное упрощение, и потому не следует придавать слишком большое значение предсказаниям на базе такой модели. Продуктивность оказывает влияние на вклад, который в свою очередь должен оказывать влияние на вознаграждение, поэтому надо разобраться с исключенными в данной модели факторами, способными существенно повлиять на продуктивность.

Затем я предполагаю, что вклад – «работа», за которую мы платим – является произведением сложности задачи на продуктивность. Иными словами, если люди одинаково продуктивны, то тот, кто решает более сложные задачи, вносит больший вклад.

Я также делаю предположение, что при данном уровне сложности задач для всех уровней квалификации, находящихся внутри канала потока, продуктивность одинакова и произвольно принята равной 1. Это представляется разумным, поскольку участники команды могут быть оптимально продуктивны в любом месте канала потока. В примере, который приводится ниже, канал потока довольно широк: это часть модели. Установка ширины канала в зависимости от других факторов – это один из способов настроить модель.

Затем я предполагаю, что при равном уровне сложности работы (горизонтальная линия на графике) продуктивность убывает по мере удаления от канала потока – как в области тревоги, так и в области скуки. В нашем примере я заставил продуктивность снижаться довольно резко. При одном уровне квалификации вне канала продуктивность падает до 0,75, при втором до 0,50, при третьем до 0,25 и при четвертом до нуля. Если выбрать другую скорость снижения продуктивности, получится иная картина. Это еще один способ «настроить» модель.

График на рис. 16.9 показывает вклад, измеренный в произвольных единицах; я помещаю в центр графика вклад в 100 единиц. Уровень сложности работы изменяется в канале потока на 10 единиц при каждом перемещении вверх или вниз. Не забывайте, что это очень упрощенная модель. И все же с ее помощью можно приблизительно оценить эффективный вклад, если теория Чиксентмихайи верна. Если мы примем эту модель,

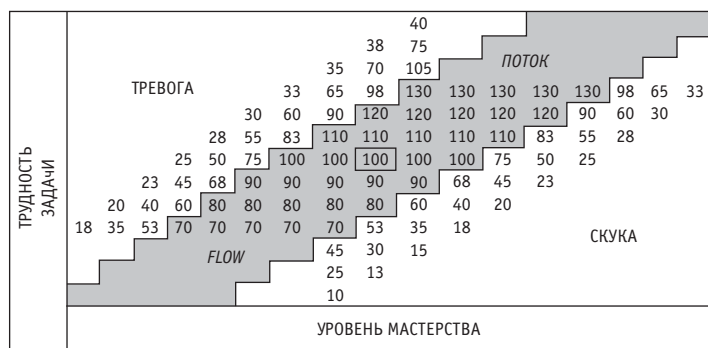


Рис. 16.9. Эффективный вклад на основе модели Чиксентмихайи

то получим представление о функциональной зависимости вознаграждения от уровня квалификации и сложности задания.

Пусть число 100, заключенное в рамочку, представляет работника. Его уровень вклада равен 100, и он находится в канале потока. Пусть ему выплачивают 100 денежных единиц за 100 единиц вклада. Сравним это с человеком, находящимся вне канала, на том же уровне сложности задачи и с вкладом 75 единиц; со значением 75 он может находиться слева или справа от канала потока. Если его вознаграждение больше 75 единиц, то согласно данной модели мы считаем, что ему переплачивают, потому что его зарплата больше, чем вклад. Или, иными словами, если человеку в канале с вкладом 100 единиц и человеку вне канала с вкладом 75 единиц одинаково платят по 85 единиц, то первому из них недоплачивают, а второму переплачивают.

Это интересный способ сравнения, который я нигде больше не встречал.

На этом часть IV завершается. Перехожу к «нестандартному мышлению».

ЧАСТЬ ПЯТАЯ

Нестандартный подход

Разобраться с предыдущими двумя частями было не так-то просто. Главы, посвященные управлению проектом и человеческому фактору, должны были заставить основательно задуматься над некоторыми тернистыми путями в нашем ремесле. Данная часть позволит читателю сделать некоторую передышку. Прежде чем перейти к крещендо в части VI «Более сложные вопросы», я порадою вас спокойной интерлюдией.

Тема этой части – разнообразие мнений, бытующих среди тех, кто занимается разработкой программного обеспечения. Некоторые из их идей спекулятивны, иногда довольно необычны. Полезно получить представление о том, что можно по-разному взглянуть на проблемы. Как спорт кто-то назвал «секцией игрушек в жизни», так и эту часть можно воспринимать как секцию игрушек в данной книге.

В главе 17 «Урок истории» я рассказываю об интересном эпизоде строительства военно-морского флота. Из гибели шведского военного судна «Ваза» в первом же выходе в море, случившейся несколько веков назад, разработчики ПО могут извлечь ряд ценных уроков.

Глава 18 «Неправильные аналогии» поднимает тему псевдонауки. Прибегая к скверным научным аналогиям, чтобы «объяснить» непрофессионалам проблемы программирования, можно столкнуться с неприятностями, поэтому данная тема достойна рассмотрения.

В главе 19 «Трудности обновления» я стараюсь по-новому осветить интересные вопросы обновления уже установленного ПО. Мое предложение по этому вопросу может оказаться для вас сюрпризом.

Наконец, в главе 20 «Числа, случайные и не очень» мы в последний раз встречаемся с Роско Леруа. Он расскажет нам о некоторых расчетах, выполненных им на необитаемом острове без помощи компьютера. Эта история о том, как анализировать задачу, которая сначала выглядит простой, но потом оказывается более сложной, чем предполагалось.

ГЛАВА СЕМНАДЦАТАЯ

Урок истории

В 1993–1994 гг. я жил в Стокгольме и работал у одного из наших самых крупных и важных клиентов. Яаак Урми (Jaak Urmi), мой друг и временный начальник, настоял на том, чтобы я провел день в Музее «Ваза» в стокгольмской гавани. Это уникальный музей.¹ Гордость Королевского флота Швеции, «Ваза», была спущена на воду в августе 1628 г. Через двадцать минут после спуска налетел небольшой порыв ветра, и она затонула на дне стокгольмской гавани. Через 333 года она была поднята на поверхность, тщательно отреставрирована и выставлена на всеобщее обозрение. Для анализа причин трагедии было отпущено несколько веков, и строительство военно-морских судов за этот период достигло значительных успехов, поэтому мы неплохо представляем себе причины, по которым затонула «Ваза». Из этого несчастного случая можно извлечь ряд интересных уроков, в том числе полезных для разработки ПО.²

¹ См. <http://www.vasamuseet.se>.

² Мы с Томом Лавом (Tom Love) независимо увидели связь между ПО и «Вазой» примерно в одно и то же время. См. Tom Love, *Object Lessons: Lessons Learned in Object-Oriented Development Projects* (New York: SIGS Books, 1993), pp. 1–6. Данный материал был изначально написан в феврале 1994 г. и опубликован в техническом бюллетене Rational под названием «What Can We Learn from the Vasa?» *The Sheep* (#40, June 1995), pp. 7–8. Он был также опубликован со всеми правами и ссылками в книге Пьера Робийяра и Филиппа Крухтена (Pierre Robillard and Philippe Kruchten, *Software Engineering Processes: With the UPEDU*, Boston: Pearson Education, 2002, pp. 151–152).

Не берите в архитекторы короля

Одна из проблем «Вазы» состояла в том, что спецификации для нее составлял и переделывал сам король. Корабль по конструкции находился посередине между «малыми» и «большими» судами того времени. Киль был первоначально заложен как для маленького судна, а потом несколько удлинен. Это помешало строителям увеличить соответствующим образом ширину судна, хотя они и пытались это сделать. В конечном счете «Ваза» получилась высоким и узким судном.

Кроме того, у нее было две орудийные палубы. В первоначальном проекте была только одна такая палуба. Но король узнал, что в соперничающей Дании строилось судно с двумя орудийными палубами, и он не хотел отстать. Позднейшее добавление второй орудийной палубы утяжелило надводную часть судна.

У программистов есть термин для такой ситуации. Это называется «клюдж» (*kludge*).

Внешнее впечатление обманчиво

Первым впечатлением, когда видишь сегодня «Вазу», оказывается изумление. Корабль высокий, узкий, с тяжелыми надстройками. Однако такую конструкцию имели все корабли той эпохи, и в большинстве своем они были устойчивы. Расчеты, проведенные по восстановленному судну, показывают, что ошибки в проекте были, но не столь значительные, как может показаться: центр равновесия приподнят всего на несколько *сантиметров*. Это достаточно удивительно.¹

Контроль проекта

Чертежей «Вазы» не сохранилось, поскольку их никогда и не было! Сегодня подобное отсутствие документации было бы скандалом, но в те времена наличие детальных чертежей не было обязательным; у кораблестроителя были тщательно оберегаемые таблицы расчетов и эмпириче-

¹ Значительная часть технической информации по «Вазе» взята из наиболее авторитетного (англоязычного) издания по этой теме *Why Vasa Capsized* (ISBN 91-85268-23-2), Курта Боргенстама и Андерса Сандстрема (Curt Borgenstam, Anders Sandstrom), изданного в 1985 г. Эта небольшая книга (80 страниц) опубликована Национальным музеем моря в Стокгольме тиражом 2000 экземпляров.

ские правила, а в основном судно строилось по вдохновению. Хотя в те времена была распространена такая практика, недопустимая сегодня, можно только гадать, могло ли наличие чертежей повлиять на последовавшие события.

Знать, чего ты не умеешь

Практика кораблестроения того времени пребывала в зачаточном состоянии. Расчет центра масс делать не умели или не хотели. Исходя из имевшихся теоретических и практических средств, было почти невозможно выяснить, устойчива ли данная конструкция. Говорить о какой-либо надежности прогнозов не приходилось. Удивительно лишь, что тонуло так мало судов.

Аналогичная ситуация сложилась в Америке со строительством мостов в начале IX века. Очень многие мосты разваливались, потому что инженеры не умели делать точные расчеты. Джон Реблинг, построивший Бруклинский мост, знал, что не умеет рассчитывать мосты, поэтому он предусмотрел повышенные меры предосторожности. Умница.

Преимственность руководства

Руководитель постройки Генрик Гибертссон умер, не успев завершить проект, и какое-то время перед его смертью руководство осуществляли вместе он и его помощник Хейн Якобссон. Оно было не очень удачным – лишнее свидетельство необходимости иметь одного сильного лидера.

Как всегда, все в спешке

«Ваза» была построена примерно за два года под сильным давлением с целью ускорить постройку. Несомненно, это повлияло на принятие ряда неудачных решений.

Сосредоточенность на несущественных деталях

Витиеватая резьба и украшения, законченные к моменту спуска, производят сильное впечатление. Очевидно, на эту работу были израсходованы большие ресурсы. На самом деле это были не просто украшения: они должны были оказать психологическое воздействие на прибалтийских со-

седей Швеции. Они бы и выполнили эту функцию, останься судно на плаву. Эта подробность – хороший пример губительного действия «ползучего фичеризма» на надежность продукта.

Если проект плох...

После катастрофы строителей критиковали за то, что они загрузили в трюмы недостаточно балласта, что позволило бы повысить остойчивость корабля. На самом деле балласта набили столько, сколько было возможно, но места для него было мало. И даже если бы оно было, более низкая осадка судна не помогла бы, поскольку при волнении вода стала бы захлестывать судно через порталы нижней оружейной палубы. Конечно, если бы была только одна оружейная палуба...

Важность тестирования

За несколько дней до спуска конструкторы осуществили очень практичную проверку устойчивости: они заставили 30 моряков перебегать с одного борта «Вазы» на другой и обратно. После нескольких попыток испытание пришлось прекратить, потому что судно слишком кренилось. Проверка закончилась зловещим предзнаменованием.

Однако короля в тот момент не было в стране, и никто не посмел отложить спуск. Что последовало дальше, всем известно.

Любопытно, что представление, будто с помощью тестирования можно «добавить» качества в программное обеспечение, столь же ошибочно. После того, как судно (или программное обеспечение) построено, тестирование может только *охарактеризовать* то, что в нем уже есть. Напротив, если проводить тесты часто и начать делать это на ранних стадиях, могут обнаружиться ошибки, которые удастся вовремя исправить. В данной ситуации едва ли такое было возможно. Однако в большинстве проектов разработки ПО раннее начало тестирования должно быть взято за *правило* в процедуре итеративной разработки.

Прототип и продукт

С учетом того, что проект «Вазы» оказался экспериментальным, судно следовало рассматривать как прототип, а не продукт. Обнаружив во время испытания неустойчивость, судно следовало сразу поместить в му-

зей – на 333 года раньше. Однако за постройкой наблюдал король, судно обошлось в 4% валового национального продукта того года, и оно было срочно необходимо для предстоявшей войны. В результате объявили, что это продукт, и запустили его.

Выводы следствия

Из 150 находившихся на борту людей погибло от 30 до 50. Сразу после трагедии оставшийся в живых капитан был арестован и посажен в тюрьму. Конечно, было расследование. Но как только знающие люди поняли, что слишком многие проблемы связаны с вмешательством самого короля, дело стало окутываться туманом. Расследование ничего толком не дало: никто не был признан виновным. Мачты, торчавшие над поверхностью воды, спилили, и вся история была тихо погребена в море.

Мы не любим копаться в провалах своих программных проектов. Я сомневаюсь, что когда-нибудь будет создан музей, посвященный какой-нибудь нашей знаменитой катастрофе.

Резюме

Инженеры-программисты могут извлечь для себя несколько полезных уроков из катастрофы «Вазы».

ГЛАВА ВОСЕМНАДЦАТАЯ

Неправильные аналогии

Опасное это занятие рассказывать о программном обеспечении и его разработке, прибегая к аналогиям. Беда в том, что аналогии могут оказаться частично верными, а частично обманчивыми или совершенно ошибочными. Например, никто не станет утверждать, что создание программного комплекса в начале XXI века сильно напоминает постройку военного корабля для шведского флота в начале XVII, и тем не менее именно к такому сравнению мы обратились в предыдущей главе. Дело в том, что из того опыта можно извлечь ценные уроки, только не надо заходить в аналогиях слишком далеко.

Аналогичную тенденцию мы наблюдаем, сталкиваясь сегодня с многочисленными примерами употребления научного языка для «объяснения» распространенных явлений. К сожалению, такое употребление часто оказывается некорректным с точки зрения метафоры и аналогии, лишенным смысла или просто глупым. В данной главе мы отобрали несколько распространенных примеров, показали недостатки приведенных в них аналогий и попытались показать, как можно лучше выразить то, что хотел сказать автор, не прибегая к жаргону, призванному произвести впечатление на непосвященных.

Хьюстон, у нас проблемы

Платон был первым философом, обратившим внимание на то, что в умозрительном постижении физического мира заключено больше

коварства, чем может показаться на первый взгляд. Из более близких нам по времени философов надо отдать должное Иммануилу Канту, попытавшемуся выяснить, какие из наших знаний могут быть получены путем «фильтрации действительности» и без нее. Речь идет о том, что мы предполагаем существование «объективной» реальности, но в некотором смысле никогда не сможем ее постичь, потому что она фильтруется тем механизмом, с помощью которого человек ее ощущает – нашими чувствами, разумом, эмоциями. Все, что мы в состоянии ощущать как индивиды и как биологический вид, – это «субъективная» реальность, и из-за данного ограничения возникает масса сомнений по поводу того, что же такое «реальность». Эти сомнения пронизывают наше мышление, вызывая иногда очень трудно уловимые последствия, подчас на подсознательном уровне. Вспомните, как часто мы повторяем, что «действительность не всегда такая, какой нам представляется».

Наши познания в физике в некотором смысле усугубляют проблему. Начиная с Галилея и Ньютона ученые выдвигали теории, которые в их время казались совершенно противоречащими здравому смыслу, – например, что движущиеся тела, предоставленные самим себе, будут продолжать двигаться вечно. По прошествии веков, однако, многие (но не все) из этих противоречащих здравому смыслу идей были приняты образованной частью общества; эти идеи стали общеизвестны и ассимилировались в качестве общепринятых доктрин. Выражаясь иначе, они стали частью нашего коллективного сознания.

Но «современной» физике всего около 100 лет. Хотя скорость, с которой население в целом принимает новые научные идеи, возросла, фактически многие из них не были усвоены. Тому есть весьма веские причины. Во-первых, «новые» теории, такие как теория относительности и квантовая механика, имеют дело с областями реальности, выходящими за границы нашего повседневного опыта, – в этих областях с их околосветовыми скоростями и субатомными расстояниями действуют законы, совершенно противоречащие даже нашему «современному» образу мышления. Как физики, так и не физики признают, что излагать эти фундаментальные теории очень трудно, отчасти потому, что математический аппарат, делающий их понятными ученому, попросту недоступен для большинства людей. Поэтому перед физиками стоит препятствие: у них есть правильные и мощные теории, но объяснить их обычным людям невозможно.

Тем не менее люди хотят разобраться. В результате популяризаторы науки, как правило, пытаются объяснять по аналогии. Это совершенно до-

пустимо. Но с течением времени сами аналогии приобретают для широкой публики статус фундаментальных истин и становятся обычными избитыми истинами. Всякий раз, когда это происходит, люди, которые, подобно мне, имеют некоторое представление о лежащих в основе научных теориях, оказываются в недоумении от такого злоупотребления неуместными аналогиями.

Я получил физическое образование и хочу сразу заявить, что не считаю, будто физика (или наука и математика в целом) является исключительно епархией неких высоких священнослужителей и что мирянину «нечего болтать о том, чего он не понимает». Я бы хотел, чтобы все лучше разбирались в науке и технологии. Но я также реалистически отношусь к огромному масштабу этой проблемы. Я был бы рад хотя бы тому, чтобы люди лучше понимали действительный смысл ученых клише и пользовались ими только тогда, когда они уместны, либо приводили вместо них более удачные аргументы. В частности, я хочу возразить против обычая щеголять научной лексикой с целью произвести впечатление на публику и убедить ее в правильности чьей-то позиции. Такая практика сродни школе доказательства, основанной на «ссылке на авторитеты». И, по моему убеждению, лучше всего это можно сделать, указав, где популярные аналогии действительно применимы, а где нет.

Итак, приступим. Попробуйте сами обнаружить, в чем заключается ошибка. Начнем с некоторых пережитков классической физики – идей, слабо понятых до сего времени, даже после многовековой варки в котле человеческого интеллекта.

Ложные законы Ньютона

Законь движения Ньютона (или три закона движения) цитируют не скупясь. Приведу некоторые примеры распространенных высказываний.

Это может сказать кто угодно:

«Этот объект находится в равновесии, поэтому согласно первому закону Ньютона на него не действуют никакие силы».

От менеджера проекта по поводу чьего-то чужого проекта:

«Этот проект явно находится в состоянии свободного падения».

От менеджера в ответ на получение отрицательной реакции на свою бизнес-инициативу:

«Этого можно было ожидать. Третий закон Ньютона говорит, что для всякого действия есть равное и противоположно направленное противодействие».

Рассмотрим последовательно эти случаи.

Ошибочное применение первого закона Ньютона

Первый закон Ньютона гласит:

Тело, находящееся в покое или движущееся равномерно (с постоянной скоростью), сохранит это состояние, если на него не окажет воздействие внешняя сила.

Обратите внимание: это означает, что на тело не действуют никакие *суммарные* внешние силы, пока об этом не будет определенно заявлено. Иными словами, на тело *могут* действовать внешние силы, но они (эти многочисленные внешние силы) полностью аннулируют друг друга. Когда эти внешние силы уравнивают друг друга, объект находится в равновесии: статическом равновесии, если тело покоится, или равновесии равномерного прямолинейного движения, если тело движется по прямой с постоянной скоростью. Поэтому запомним: равновесие не означает «отсутствие сил». Равновесие означает, что «все внешние силы в точности уравнивают друг друга». Конечно, *внутренние* силы воздействия не оказывают, потому что попарно уничтожают друг друга в соответствии с третьим законом Ньютона, о чем мы вскоре скажем.

Предположим, что кусок угля движется по поверхности горизонтального стола с постоянной скоростью. Отвлечемся пока от того, почему он начал двигаться, и предположим, что он перемещается в западную сторону на один дюйм в час. Первый закон Ньютона утверждает, что если мы не приложим к этому куску какую-нибудь горизонтальную силу, он будет *вечно* перемещаться на запад по одному дюйму в час.

Как я уже отмечал выше, это противоречит здравому смыслу. В реальном мире следует ожидать, что кусок угля будет замедлять движение и в конце концов остановится, как минимум, по двум причинам. Во-первых, из-за сопротивления воздуха, во-вторых, из-за трения о поверхность стола. Оба эти фактора замедляют равномерное движение на запад. Но никакого нарушения первого закона Ньютона здесь нет: сопротивление воздуха и трение – это внешние силы, действующие на кусок угля, а первый закон явно указывает, что правило *неприменимо*, если на данное тело действуют внешние (суммарные) силы. Физик, привыкший к точному мышлению и фор-

мулировкам, понимает, что сила есть сила, и пренебрегать ими нельзя. Точно описать данную ситуацию следовало так: «Кусок угля будет бесконечно долго перемещаться на запад по одному дюйму в час в условиях абсолютного вакуума по абсолютно горизонтальному столу в отсутствие трения». Беда в том, что мы в большинстве своем не описываем повседневные явления с такой точностью, поэтому легко представить себе, как обычный человек может неправильно применить первый закон Ньютона.

Один из более молодых физиков недавно обратил мое внимание на то, что современные студенты рассматривают удаленное космическое пространство как подходящую среду для решения задач, в которой можно легко избавиться от влияния сопротивления воздуха, трения, «столков» и гравитационного притяжения близких массивных тел. В таком идеализированном контексте упрощаются требования для понимания механики, но остается только гадать, что произойдет, если этим студентам придется решать реальные задачи «по возвращении на Землю».

Ошибочное применение второго закона

Второй закон гласит:

Под действием внешней силы тело получает ускорение, прямо пропорциональное этой силе и обратно пропорциональное своей массе.

Его часто записывают формулой $F = ma$, но это лишь другой способ записи.¹ Это невероятно простой и элегантный результат, применимый к чрезвычайно широкому кругу явлений.

Но какой смысл вкладывается в фразу о «свободном падении проекта»? Я думаю, что менеджеры имеют в виду его ускорение под действием силы тяжести, в результате которого он наберет скорость и неизбежно совершит неупругое и катастрофическое столкновение с матушкой землей. Шмяк! Я так понимаю, что парашюты или тормоза отсутствуют, отсюда и ощущение быстро приближающейся трагедии. Отмечу, однако, что обращение к физической аналогии здесь неуместно. У проектов есть ограничения,

¹ Обычно приводится формула $F = ma$, но более общая формула выглядит как $F = dp/dt$, что означает: сила пропорциональна скорости изменения количества движения. Это существенно только тогда, когда масса системы непостоянна, как в случае ракеты, которая сжигает топливо и теряет массу во время полета. Выражение $F = dp/dt$ более общее, чем $F = ma$, но последнее встречается чаще. Просто не забывайте, что последняя формула содержит предположение о том, что масса не меняется.

и несомненно, что у них есть масса (инерция). Представление о том, что руководство настолько самоустранилось от работы, что мы фактически выдернули стол из-под нашего куска угля, как минимум вызывает уныние.

Ошибочное применение третьего закона

Третий закон гласит:

При взаимодействии двух тел сила, с которой первое тело действует на второе, равна и противоположна по направлению силе, с которой второе тело действует на первое.

Когда в мире бизнеса что-то происходит, кто-нибудь обязательно проблеет: «каждое действие вызывает равную и противоположную реакцию». На самом деле это у него самого подколенная «реакция». Вместо того чтобы немного поразмышлять над ситуацией, он цитирует Ньютона для оправдания или обоснования отрицательного отношения рынка. О реакции говорят как о неизбежности, следующей из законов физики. На самом деле, однако, все происходящее не имеет никакого отношения к физике. Обычно реакция не только не равна тому действию, которое ее вызвало, но и направлена не в обратном направлении, а под некоторым углом. Кроме того, она может и вовсе не быть результатом первоначального действия.

Опять-таки закон Ньютона верен, но необходимо точно указать тело и силу. Часто «равные и противоположные» силы, о которых говорят в деловых ситуациях, в действительности представляют собой пару внутренних сил и не порождают *никакой* суммарной внешней силы, прилагаемой к телу. Поэтому когда кто-то произносит заклинание «каждое действие вызывает равную и противоположную реакцию», советую проверить, что за силы присутствуют и к каким телам они приложены.

Все относительно

Теперь, разобравшись с опасностью неправильного толкования Ньютона, проверим еще пару популярных идиом. Два человека в чем-то не согласны между собой, и один из них заявляет:

«Ну, все зависит от выбранной системы отсчета».

Либо некто хочет отметить, что «считавшиеся прежде справедливыми законы природы больше неверны». Обычно для этого употребляют выражения типа

«Эйнштейн показал, что Ньютон ошибался».

Как говорится в рекламе Hertz, это не совсем точно. Эйнштейн проделал гигантскую работу, создав теорию относительности, но эта теория сама породила некоторые странные представления.

Система отсчета

Что касается первого примера, то действительно, картина может быть разной в зависимости от перспективы или точки зрения наблюдателя. Но, как впервые отметил Галилей, реальность *не* меняется в зависимости от выбора системы отсчета. Находясь в системе, которая движется с постоянной скоростью, нельзя узнать, какой кажется эта скорость неподвижному наблюдателю, потому что все в вашей системе ведет себя по законам физики и выглядит точно так же, как если бы вы были неподвижны. Работа Эйнштейна объяснила видимое противоречие между принципом Галилея и электродинамикой Максвелла. Аналогичным образом нельзя различить ускорение и искривление пространства-времени.

Эйнштейн опроверг Ньютона?

Что касается второго заявления, то законы Ньютона совершенно верны при скоростях, с которыми мы имеем дело в повседневной жизни. Положение изменяется только при движении со скоростями, приближающимися к скорости света. Тогда потребуются другие законы. Тут-то и вступит в игру теория Эйнштейна. Законы Ньютона действуют на малых скоростях (т. е. в большей части случаев), а теория относительности Эйнштейна вступает в силу, когда вы начинаете двигаться очень быстро. Если слишком долго полагаться «только на Ньютона», то *с приближением* к скорости света будут получаться все более некорректные результаты, а с достижением скорости света ответы станут *совершенно* неправильными. Надо только помнить, что по сравнению с нашим повседневным опытом скорость света представляет собой невероятно большую величину.

Эффектами теории относительности в повседневной жизни можно абсолютно пренебречь. При желании можете рассчитать их, воспользовавшись специальной теорией относительности Эйнштейна, но вы обнаружите, что ваши результаты несколько не меняются.

Дело в том, что скорость света очень велика. Напротив, со скоростью звука мы сталкиваемся ежедневно. Можете проверить, насколько скорость света отличается от скорости звука, проделав такой эксперимент, когда будете в следующий раз играть в гольф. Пройдя метров 250 по фарватеру (уже сде-

лав второй удар и двигаясь дальше), оглянитесь и посмотрите (и послушайте), как следующая группа делает первые удары. Вы увидите, как клюшка ударяет по мячику, а через долю секунды слышен удар. Можете рассчитать этот интервал, воспользовавшись значением скорости звука в сухом воздухе на уровне океана¹ и предположив, что скорость света бесконечна, т. е. свет проходит путь от клюшки для гольфа до ваших глаз за нулевое время. Вы должны получить правильный ответ.² Если же вы выполните расчет с учетом реальной скорости света, то получите в принципе тот же самый ответ.³ Поэтому, хотя вы с полным основанием можете применить здесь теорию относительности, задав конечную скорость света, вы не много от этого выиграете.

Тем не менее в реальной жизни можно ощутить конечность скорости света. При разговоре по телефону с другой страной вам может не повезти, и ваш звонок сначала будет передан на геостационарный спутник, а потом обратно на Землю, что займет примерно полсекунды. Это достаточно длинный интервал, который может создать ощущение, что ваш собеседник делает паузы. Эту задержку вы можете ошибочно истолковать как его несогласие, колебание, опасение и т. д. в зависимости от содержания разговора.

И еще одно...

Заговорив об Эйнштейне, мы коснулись его теории очень поверхностно. Все рассмотренные нами явления служат проявлениями его специальной теории относительности, которая справедлива только для тел, движущихся с постоянной скоростью. Для тел, движущихся с релятивистским ускорением, необходимо применять его общую теорию относительности и вместе с нею мощный дополнительный математический аппарат. Однако популяризаторы столь же безнаказанно ссылаются и на общую теорию относительности. В действительности существует лишь очень небольшое

¹ Обратите внимание, как наш язык становится точным, когда мы хотим аккуратно пользоваться законами физики. Скорость звука зависит от множества параметров, в том числе температуры, о которой я ничего не сказал.

² Надеюсь, вы оказались достаточно любопытны, чтобы самостоятельно проделать вычисления. Если нет, то я их приведу. При 68 градусах Фаренгейта скорость звука составляет 1127,3 фута в секунду. 250 ярдов – это 750 футов, поэтому звук достигнет вас через 0,665, или примерно через $2/3$ секунды. Это ощутимый промежуток времени. Кстати, я воспользовался британской системой единиц, а не метрической, поскольку игроки в гольф обычно «калиброваны» в ярдах, а не в метрах.

³ Другой стандартный пример такого расчета – определение расстояния до точки удара молнии по разнице времени между вспышкой и громом. Принцип тот же.

число экспериментальных проверок общей теории, и во всех из них участвуют крайне малые эффекты.¹

Это нисколько не уменьшает величие достижений Эйнштейна. Однако применение его блестящих открытий в тех случаях, когда в этом нет необходимости, несколько принижает их значение.

Квантовая чушь

Перейдем от теории относительности к квантовой механике. Недавно мне довелось услышать от человека, не желающего делать прогноз, такие слова:

«Это как в квантовой механике. Я могу сообщить вам лишь значение вероятности».

Хотя вторая часть его заявления была, конечно, верна, у меня нет сомнения в том, что она никак не связана с квантовой механикой.

Примерно через 20 лет после революции, произведенной теорией относительности, т. е. в 1927 г., квантовая механика обрушилась на человечество, оказав такое же важное и тревожное влияние.²

О квантовой механике следует помнить все то же, что и о теории относительности: ни одна из этих теорий не отменяет законы Ньютона. Теория относительности Эйнштейна развивает законы Ньютона для области сверхвысоких скоростей (приближающихся к скорости света), а квантовая механика развивает классическую физику для области сверхмалых расстоя-

¹ Когда лет 30 назад я занимался физикой, таких проверок было известно всего три. Это были прецессия перигелия орбиты Меркурия, гравитационное искривление света при прохождении около массивного объекта и гравитационное смещение света в красную сторону при выходе из гравитационного поля массы. Согласно имеющимся у меня сведениям с тех пор появилось еще несколько экспериментов, в одном из которых проводятся измерения на двойных пульсарах. До недавнего времени все эти эффекты были крайне малы, с трудом поддавались измерению и имели очень слабое отношение к повседневной жизни. Однако сегодня часы на спутниках системы GPS должны корректироваться для учета эффектов, обусловленных исключительно общей теорией относительности, поэтому даже эту область технология ввела в нашу повседневную жизнь.

² Некоторые связывают дебют квантовой механики с работами Планка в первые десятилетия XX века, проводившимися одновременно с исследованиями Эйнштейна. Я указываю 1927 г., поскольку в начале именно этого года были опубликованы статьи Шредингера, написанные в 1926 г. и давшие нам уравнение Шредингера. Они формализовали ситуацию и послужили подлинно революционным толчком.

ний. Когда мы достигаем субатомных размеров, начинают действовать новые правила. Вот тогда нам понадобится квантовая механика. Во всех остальных ситуациях законы квантовой механики сохраняют силу, но их эффект настолько мал, что им можно пренебречь.

Важно понимать, что квантовая механика – это одна из самых успешных теорий всех времен. Она смогла объяснить широкий круг противоречащих интуиции явлений, которые мы теперь стали воспринимать как сами собой разумеющиеся. Без этой теории мы не знали бы, как работают полупроводники, и тогда использование мною процессора Pentium для написания этой книги оказалось бы под вопросом. Она непосредственно объясняет причину стабильности атомов, без которой, как отмечает Джон Уокер, весь день идет насмарку. Поэтому, ежедневно имея дело с технологиями, основанными на квантовой механике, мы редко наблюдаем явления, в которых непосредственно проявляются квантовые эффекты. Это достаточно тонкий момент.

Причина, по которой потребовалось так много времени, чтобы открыть эти две области знания, заключается в том, что мы не умели измерять очень быстрые процессы и очень маленькие объекты вплоть до второй половины XIX века. Фактически измерения в обеих областях стали возможны с изобретением и усовершенствованием вакуумного насоса, что было подвигом в инженерном деле. Это также объясняет, почему не наблюдались явления, требовавшие для своего объяснения теории Эйнштейна или квантовой механики; за исключением загадки корпускулярно-волнового дуализма света, известной еще во времена Ньютона, ничто в нашем медлительном макром мире не указывало на то, что «не все в порядке».

Цитата «это как в квантовой механике» служит примером еще одного интересного заблуждения. Поскольку квантовая теория основана на исчислении вероятностей, многие считают, что выводы, сделанные на основании квантовой механики, являются не вполне точными. В действительности все обстоит ровно наоборот.

Например, мы можем экспериментально определить значение константы тонкой структуры α .¹ Это число оказывается квинтэссенцией «со-

¹ Константа тонкой структуры возникает при изучении расстояния между линиями в спектре атомов элемента. Квантовая механика появилась в результате попыток физиков объяснить различие расстояний для разных элементов. Позднее с помощью квантовой теории были предсказаны спектральные эффекты более высокого порядка, например при воздействии на атомы электрического или магнитного поля.

временной» физики: среди прочего оно зависит от заряда электрона, постоянной Планка (подробнее о ней ниже) и скорости света. Каким бы способом вы ее ни измеряли, вы прибегаете к квантовой механике, и в ее теоретическом предсказании участвует ряд самых глубоких приложений квантовой теории. Так вот, можно поставить эксперименты и измерить эту константу с точностью до одной части на 10^8 . Это довольно хороший результат для любых приложений.

Для сравнения: универсальная гравитационная постоянная G – это абсолютно «классическая» величина, известная со времен Ньютона, и она экспериментально измерена только с точностью до одной части на 10^4 . Это тоже неплохой результат, соответствующий точности 0,01%. Тем не менее α известна нам с точностью, в несколько тысяч раз *более высокой*. Не правда ли, в этом есть ирония?

Вот такова вероятностная природа квантовой механики и ее возможности предсказания.

Еще некоторые квантовые глупости

На самом деле моя большая мозоль – частое упоминание все «принципа неопределенности Гейзенберга». Если вам нужен самый необычный пример, послушайте монолог Фредди Риденшнайдера из фильма братьев Коэнов «Человек, которого не было» (The Man Who Wasn't There).¹ Вполне понятно, почему Билли Боб Торнтон получил электрический стул, когда его адвокат попытался использовать этот принцип, чтобы убедить (или запутать) жюри присяжных.

Когда кого-то просят провести сложное измерение, можно услышать стандартную жалобу:

«Ничего не выходит. Гейзенберг учит нас, что нельзя измерить что-нибудь, не внося возмущений».

А вот еще пример. Программисты изобрели слово «гейзенбаг».

«Старик, мы несколько недель искали ошибку. Оказалось, что это гейзенбаг».

Так называют «баги», которые очень трудно исправить, потому что при попытке сделать это изменяется работа программы и исходная ошибка прячется еще глубже из-за действий самих средств отладки.

¹ USA Films, 2001.

Что же здесь происходит на самом деле?

Измерения

Вот фундаментальная проблема: можно ли измерить что-либо, не повлияв при этом на измеряемый объект? Иными словами, влияет ли каким-то образом процесс измерения на его предмет? Если да, то возникает проблема, поскольку измерение будет искажено в результате возмущения системы, которую требуется измерить.

Нельзя сказать, что это очень «сложная» проблема. В медицинской диагностике приходится постоянно иметь с ней дело. Медики тратят массу времени и сил, чтобы сделать процедуры диагностики минимально агрессивными. Тем не менее известно, что у некоторых людей кровяное давление поднимается в тот самый момент, как им на руку одевают манжету для его измерения. Следовательно, результат измерения для них будет выше, чем их нормальное давление в состоянии покоя.

В программировании мы очень стараемся создать такие отладчики, которые не мешают работе программы. Тем не менее иногда при отладке *что-то* меняется, и программа начинает вести себя не так, как в отсутствие отладчика. Чья тут вина – программы или отладчика – вопрос неясный, но в любом случае у программиста возникают большие трудности.

В третьем десятилетии XX века Элтон Майо открыл *эффект Готорна*. Он продемонстрировал, что при изучении человеческого поведения трудно разделить изучаемое поведение и те изменения, которые неизбежно происходят, когда изучаемой группе известно, что она подвергается изучению. По этой причине сегодняшние медицинские эксперименты проводятся двойным слепым методом: ни пациент, ни врач не знают, кто получает лекарство, а кто плацебо.

Обратите внимание на абсолютно «классический» характер этих явлений: для их объяснения не требуются ни квантовая механика, ни принцип неопределенности Гейзенберга.

Прежде чем глубже заняться Гейзенбергом, зададимся таким вопросом: можно ли провести хотя бы одно какое-то измерение, которое будет абсолютно «неинтрузивным»? Если я найду лишь один такой пример, то опровергну предположение о том, что это невозможно.

Вот мой пример. Я просыпаюсь на больничной койке в комнате, которой никогда раньше не видел. Я хочу узнать размеры этой комнаты. Для этого я считаю плитки на потолке. Потолок имеет 16 плиток в длину

и 12 плиток в ширину. Я знаю, что плитки имеют стандартный размер фут на фут. Отсюда я получаю, что комната имеет размер 16 футов на 12 и площадь 192 квадратных фута. Есть! Я выполнил измерение, даже не вставая с кровати, и я утверждаю, что при этом комната нисколько не изменилась.¹

Применение принципа Гейзенберга

Область применения принципа Гейзенберга – атомный и субатомный уровень. Его суть в утверждении, что с точки зрения квантовой механики невозможно указать одновременно положение частицы и ее количество движения с произвольной точностью. Если требуется с большей точностью знать положение частицы, то ее количество движения будет определено с меньшей точностью, и наоборот.

Чтобы наблюдать упомянутую частицу, необходимо «осветить ее». Но если это сделать, то свет изменит импульс частицы, и потому узнать точное положение частицы не удастся. Поэтому на квантовом уровне измерение «без вмешательства» невозможно, и Гейзенберг вывел формулу, по которой можно вычислить, в какой степени вмешательство средства измерения изменит результат измерений.

Предупреждение: в формуле Гейзенберга участвует постоянная Планка, которая очень и очень мала. Настолько мала, что результаты действия принципа Гейзенберга неощутимы, если исследуются большие объекты, размеры которых не сравнимы с атомными и субатомными расстояниями. Это означает, что если осветить электрон, то влияние на него окажется ощутимым. Если же свет упадет на плитки потолка в больничной палате, это никак не повлияет на них. Поэтому применять принцип Гейзенберга к макроскопическим объектам просто бессмысленно.

Тепловая смерть

Переходим к последнему случаю неправильного употребления. Некоторые считают, что в физике было всего четыре или пять эпохальных переломов. Между Ньютоном и чудовищами XX века – теорией относи-

¹ Если вы строго следите за семантикой, то можете сказать, что я выполнил *оценку*, а не *измерение*. Замечу в ответ, что любое измерение оказывается оценкой, поскольку содержит некоторый элемент неопределенности. Например, если вы измерите потолочную плитку линейкой и установите, что она имеет размер «12 дюймов на 12 дюймов», будете ли вы утверждать, что это ее точный размер?

тельности и квантовой механикой – втиснулась наука под названием «термодинамика».¹

Термодинамика действительно все перевернула. От нее идут современные представления о сохранении энергии. Она делает понятной связь между работой и теплотой. Она доказывает невозможность создания вечного двигателя. Но самым интригующим является совершенно новое понятие, пришедшее из этой науки, а именно *энтропия*.

В результате сегодня можно услышать заявления такого типа:

«Крупные компании обречены на провал, потому что энтропия неизбежно побеждает».

Энтропия – это мера беспорядка. И одна из основных догм термодинамики – неуклонный рост энтропии. Пример, который часто приводят студентам, интуитивно очень привлекателен. Возьмем ящик, в середине которого есть перегородка, и наполним одну его часть молекулами кислорода, а другую – молекулами азота. Уберем перегородку, и молекулы станут хаотически перемещаться. Через некоторое время в коробке окажется равномерная смесь кислорода с азотом. Можно ждать вечно, но *никогда* молекулы по собственной воле не вернутся в то состояние, когда с одной стороны был кислород, а с другой – азот.² Состояние перемешивания считается более «случайным» или неупорядоченным; система с разделением – более упорядоченной. Энтропия образовавшейся системы выше, чем первоначальной.

В замкнутой системе энтропия спонтанно и естественным образом возрастает. В конечном итоге система достигает максимальной энтропии, или

¹ Можно отметить попутно, что примерно в период Гражданской войны в Соединенных Штатах наш друг Джеймс Кларк Максвелл, основываясь на предшествующих эмпирических трудах Майкла Фарадея, переработал электромагнетизм, придав ему чудесную математическую формулировку. В результате электричество и магнетизм стали разными сторонами одной теории, и это поразительно. В известном смысле благодаря этому смогли появиться сегодняшние средства связи; например, после этого появился Маркони и изобрел радио. Поэтому значение этой работы нельзя переоценить. И все же для меня уравнения Максвелла – это проявление *силы* математики: во времена Максвелла уже хорошо разбирались в физике и феноменологии. Например, гражданская война прервала прокладку трансатлантического кабеля, так что телеграф существовал задолго до уравнений Максвелла.

² Можно теоретически посчитать вероятность такого события. Поверьте мне, она очень близка к нулю.

полной случайности. Применяя это явление к вселенной в целом, физики говорят о ее «тепловой смерти», откуда и взято название этого раздела.

Кажется логичным, и в целом так оно и есть. Большинство систем, предоставленных самим себе, стремится к более беспорядочному состоянию. Достаточно взглянуть на мой рабочий стол.

Но в какой-то момент обыватели начали распространять это понятие на экономические и общественные системы. И мне кажется, что тут было сделано движение в неверном направлении. Обратите внимание, что в приведенной цитате есть рациональное зерно. Несомненно, что чем крупнее организация, тем большее количество коммуникационных каналов она должна поддерживать; как много лет назад отмечал Кеннет Эрроу,¹ это в конечном итоге может ограничить ее рост. Несомненно, что в больших организациях труднее координировать деятельность и даже труднее быстро реагировать на изменение обстоятельств. Однако было бы ошибкой предполагать, что энтропия неизбежно должна победить.

Замкнутая система, предоставленная самой себе, стремится к состоянию максимальной энтропии, и это совершенно верно, но экономические системы, такие как компания, в которой вы работаете, не являются «замкнутыми». Они открыты потокам вещества и энергии. И обычно мы не оставляем свои компании в изоляции. Мы постоянно пополняем их запасы исходных материалов; мы работаем в системе; мы тратим энергию на борьбу с энтропией. Так же, как я стараюсь расчистить свой стол и навести на нем порядок (уменьшить энтропию), я могу вложить свой труд в развитие каналов и механизмов связи своей компании, чтобы уменьшить беспорядок.

Эта работа примерно равноценна трате машиной энергии для преодоления трения: она в некотором смысле не является «полезной». С другой стороны, она снабжает нас мотивировкой для продолжения человеческой деятельности. Выбрав правильное соотношение, мы можем хотя бы сдерживать увеличение энтропии, пока движемся в сторону «реальных» целей. Один давно забытый философ отмечал, что мы всю жизнь занимаемся тем, что боремся с энтропией. Это способ выживания отдельных людей и общества. Фактически общественные организации – страны или предприятия, – которые лучше противодействуют росту энтропии, в конечном итоге одерживают верх над теми, кто менее успешен в этом фундаментальном занятии.

¹ Arrow, Kenneth, *The Limits of Organization*, New York: W.W.Norton & Company, 1974.

Главный вопрос, который надо задать тому, кто ссылается на неизбежность увеличения энтропии и беспорядка, должен быть таким: а замкнута ли эта система? Если нет, то не факт, что энтропия будет спонтанно и неизбежно расти.

Другие примеры

К сожалению, есть масса других примеров, которые я мог бы разбирать. На каждый ушло бы несколько абзацев, а эта глава итак уже оказалась большой. Мне доводилось слышать многочисленные ложные утверждения, касающиеся двойственной (корпускулярно-волновой) природы света. Открытия последних 40 лет, сделанные в теории хаоса, некорректно цитируются в новых попытках опровергнуть законы Ньютона. Теорема Гёделя о неполноте, которой уже лет 70, иногда эксплуатируется теми, кто пытается оправдать нашу неспособность что-нибудь доказать. В сфере вычислительной техники машину Тьюринга часто используют для доказательства неразрешимости¹ там, где она не имеет вообще никакого применения. Покойный ученый Стивен Джей Гулд много писал о том, что часто ссылаются на теорию эволюции Дарвина, в целом не понимая ее. Все эти фундаментальные теоремы носят глубокий характер, и все они дискредитируются теми, кто прибегает к ним в ситуациях, где они абсолютно неприменимы.

Хорошая наука

Ученые и математики дали нам ряд чрезвычайно мощных средств, помогающих понять окружающий нас физический мир. Эти средства являются собою замечательный триумф человеческого разума, позволяя нам, начав с самых основ, объяснить широкий круг явлений вплоть до существования и поведения элементарных частиц. Однако по мере того как эти явления удаляются от круга наших повседневных впечатлений, теории становятся все более абстрактными, и для их изложения требуется все более изощренная математика. И тогда наши достижения в понимании фундаментальных законов природы становятся жертвой непонимания широкой публикой. И хотя среднестатистический Джо² не может вполне оце-

¹ Понятие неразрешимости означает невозможность написать компьютерную программу, которая определит результат проблемы или класса проблем.

² Не исключая присутствующих.

нить все тонкости, он несомненно выгадает от результатов этих открытий, которые будут воплощены в предметах, используемых им в повседневной жизни. В данном смысле это все «хорошая наука».

А вот что *не* будет хорошей наукой, так это употребление «научных» слов при объяснении вещей, явно не связанных с физическими принципами, лежащими в основе этих слов. Люди – это *не* элементарные частицы, и нет оснований считать, что, будучи макроскопическими объектами, они подчиняются законам квантовой механики. Такие аналогии ошибочны, и следует опасаться тех, кто с их помощью доказывает правоту своих позиций.

Подобным же образом следует стараться не употреблять псевдонаучный жаргон в повседневном общении с другими людьми. Самым безвредным из последствий оказывается бездумное доверие окружающих, «потрясенных» вашим техническим лексиконом. Хуже, если они будут согласно кивать в ответ, в тайне сочтя вас просто ярмарочным шарлатаном. И вам будет казаться, что вы завоевали доверие, тогда как в действительности вы его потеряли.

Резюме

Эта глава посвящена моему дорогому другу Марку Сэдлеру (1945–2002), который покинул этот мир после мужественной борьбы с ALS (боковой амиотрофический склероз). Марк учился в Оксфорде, и колледж (Баллиол), который он посещал, вдохновлял его «превосходить без усилий». Во многих отношениях это ему удавалось. Мы часто обсуждали темы, затронутые в этой главе, и он поощрял меня шире распространять эти идеи.

ГЛАВА ДЕВЯТНАДЦАТАЯ

Проблема обновления



Мы нередко тратим неоправданно много времени, обсуждая разработку нового ПО. Хотя люди предпочитают работать именно над такими проектами, разрабатывая целину, суровая правда жизни заключается в том, что программное обеспечение живет очень долго. Вследствие этого мы тратим массу времени на сопровождение и модернизацию старого кода. Не так уж редко срок жизни систем измеряется не годами, а десятилетиями. Может возникнуть вопрос, почему программы оказываются такими живучими, если вся отрасль развивается так быстро?

Тому есть ряд причин. Во-первых, стоимость начальной разработки обычно намного превышает прогнозы, поэтому «скрытая цена» вновь развертываемой системы перекрывает ожидания уже в начале проекта. В каждой последующей точке развития проекта, где принимается решение, необходимо взвесить стоимость модернизации с одной стороны и стоимость замены в результате покупки или переписывания системы с другой. Как правило, стоимость обновления невелика по сравнению со стоимостью замены, а небольшие изменения функциональности обычно вполне осуществимы, если исходная архитектура была надежной. Количество ошибок в целом предсказуемо.

Поговорим о частоте возникновения ошибок подробнее. В начале жизненного цикла нового программного продукта частота выявления ошибок обычно относительно высока. Это те ошибки, которые проскользнули через тестирование и были обнаружены первой волной покупателей продукта. Эти ошибки можно сравнить с «младенческой смертностью» отка-

зов аппаратуры.¹ Они обнаруживаются и удаляются из системы относительно быстро.

Затем, как и в случае с аппаратным обеспечением, наступает относительно долгий период, когда ошибок и отказов мало. Для аппаратуры мы связываем такое спокойствие с тем, что вышли из строя все ненадежные детали. Если речь идет о программах, этот период соответствует зрелому продукту, схемы применения которого относительно устоялись. В основном работают одни и те же участки кода, а ошибки выявляются в тех фрагментах кода, которые редко вызываются.

Наконец, приближается «конец жизненного пути» продукта. Для аппаратуры это частые отказы, вызванные износом деталей. Это почти зеркальное отражение проблемы младенческой смерти: смерть наступает от старости. В ПО наблюдается аналогичная проблема: в старых системах частота ошибок снова растет. Это вызывается двумя причинами:

- Во-первых, в коде начинают исследоваться статистически редкие пути, и ошибка, присутствовавшая в них всегда, вдруг всплывает на поверхность благодаря необычным, прежде не встречавшимся стечениям обстоятельств.
- Во-вторых, проявляется кумулятивный эффект многолетнего сопровождения продукта.² Заплатки и исправления, сделанные поколениями программистов, настолько ухудшили первоначальную архитектуру, что код «рассыпается». Еще одно исправление, и весь карточный домик может рухнуть. Вот в такой момент замена всей системы и технически и экономически более оправданна, чем всего один лишний цикл обновления.

Суммарная стоимость жизненного цикла большинства программных систем оказывается значительной по сравнению со стоимостью первоначальной разработки. Но даже простую оценку редко проводят, когда задумывается новая система. Правильной мерой стоимости всякой новой системы должна быть суммарная стоимость всего ее жизненного цикла, но эту цифру мы редко видим на этапе предложения. Это еще одно свидетельство недостаточной зрелости нашей дисциплины.

¹ В русскоязычной теории надежности аппаратных средств принят термин «приработка на отказ». – *Примеч. науч. ред.*

² Известный и многократно описанный эффект: устранение одной явной ошибки часто вносит 2–3 новые, но более «тонкие» и скрытые, которые проявляются гораздо реже и при редком стечении обстоятельств, например особом сочетании входных данных. – *Примеч. науч. ред.*

Обновление встроенного ПО

Убедившись в том, что программное обеспечение живет долго, мы сталкиваемся с проблемой его обновления. Особенно сложна эта задача для встроенных программ. В оставшейся части этой главы я займусь неприятной проблемой обновления ПО для определенного класса устройств: карманных, мобильных, беспроводных и облегченных. К этому классу относятся сотовые телефоны, личные органайзеры, приемники GPS, цифровые камеры и различные их комбинации. Все они обладают встроенным ПО, программируемой памятью того или иного объема и аккумуляторами. Они могут обмениваться данными с другими устройствами или работать автономно.

Важность этого класса, как и количество входящих в него устройств, неуклонно возрастает, чему есть минимум две причины. Прежде всего, все продукты, предназначенные для ношения с собой, постоянно уменьшаются в размерах, и по мере того как удобство предоставляемых ими функций (например, возможность передачи цифровых фотографий) получает широкое признание на рынке, уменьшение их размеров делает их еще более привлекательными. Во-вторых, мы постоянно ищем пути освобождения от ненавистных нам пут. Беспроводная связь прекрасно способствует этому, т. к. освобождает от мучений с проводными телефонными линиями, а аккумуляторы – от других пут – проводов электрического питания.

По моему представлению, обновление ПО для таких устройств следовало сделать как можно более простым. Ниже я предлагаю теоретическое решение, но оно, как мне кажется, заслуживает дальнейшего обсуждения.

Современная ситуация

Когда сегодня вы покупаете один из таких продуктов, вам *кажется*, что вы покупаете устройство. На самом деле, вы видите в первую очередь оболочку устройства, а все самое главное заключается в ПО. Каждый программист, например, знает, что после включения сотового телефона надо подождать, пока произойдет «начальная загрузка». А кнопки с цифрами для набора телефонного номера служат также клавишами для «программирования продукта». Я имею в виду, что это один из способов записать различную информацию (например, телефонные номера) в память телефона.

В сущности устройство состоит из трех частей:

- аппаратной (hardware)

- встроенного ПО (embedded software)
- аккумуляторов, без которых, спешу добавить, ничего работать не будет

Сейчас такое устройство оформляется в виде двух составных частей: телефона, в котором находятся как аппаратная, так и программная части, и аккумуляторов. Так сложилось исторически.

При этом возникает интересная дилемма.

Весь интеллект заключен в программном обеспечении. Поэтому есть два варианта обновления телефона:

- Обновить ПО, для чего требуется заменить или перепрограммировать микросхему, находящуюся в телефоне.
- Если есть деньги, купить новый телефон, который может оказаться той же самой оболочкой, только с более новой версией микросхемы.

Для других устройств путь несколько отличается: можно подключить их к своему компьютеру, зайти на веб-сайт производителя и загрузить с него новую версию ПО.¹ Эта операция, которую иногда называют «заменой масла», позволяет заменить ПО его более новой версией. Некоторых людей аналогия с заменой масла несколько пугает, потому что это означает, что ПО может автоматически заменяться через регулярные промежутки времени, а при этом даже не будет известно, какие изменения внес производитель устройства, и такое положение вызывает у них тревогу. Другим же вся процедура обновления с помощью компьютера больше напоминает замену даже не масла, а двигателя, т. е. они вообще не считают эту задачу простой.

Игры вокруг обновления ПО

Поставщики ПО вынуждены постоянно совершенствовать свой продукт и поэтому они достаточно регулярно выпускают обновления своих программ. Те, кто покупают новое устройство, получают вместе с ним самую новую и лучшую версию. Чтобы распределить расходы на разработку и удовлетворить уже существующих клиентов, поставщики стараются побудить уже имеющихся пользователей приобрести новейшую версию ПО тоже, хотя бы и за умеренные деньги.

Как же добиваются того, что пользователи обновляют ПО?

¹ Так же выглядит и общеизвестная процедура обновления версии BIOS всех современных моделей настольных компьютеров. – *Примеч. науч. ред.*

Индустрия ПО бьется над решением этой проблемы. Люди склонны привыкать к тому ПО, которое у них есть, и заставить их заменить его самой свежей версией не так-то просто. Требуется убедить их раскрыть свой кошелек и потратить свои кровные на замену того, что и так работает неплохо. Производители испробовали много механизмов, в большинстве своем основанных на модели подписки; например, профилактика оборудования в TiVo и обновление в он-лайне AOL основаны на том, что оплата пользователями ПО (и неявно обновлений) входит в ежемесячную стоимость услуги.

Было отмечено, что крупные поставщики ПО проверяют самые различные варианты ценовой политики, включая абонентскую плату, чтобы гарантировать себе возможность как можно дольше получать устойчивые доходы.

Но если оставить в стороне экономическую компоненту, то остается техническая: как сделать обновление простым и дружественным по отношению к пользователю?

Скрамное предложение

Все, что нам нужно для сотовых телефонов, как и для всех устройств со встроенным ПО, – это *ключать его в один комплект с батареей*, а не с устройством. Вы покупаете базовое устройство без батарей или ПО, хотя пользы от него будет мало – даже в качестве упора для двери, поскольку весу в нем никакого. Но зато цена его может быть снижена до минимума, допускаемого эффективностью производства.

Затем, чтобы включить его, надо купить батареи.¹ У каждого устройства должны быть батареи своего типа. Тут же вместе с батареей вы получаете ПО. Подробности технической реализации я здесь опускаю. Можете считать, что вы приобретаете «комплект с батареями». Для маркетинга я выбрал бы название «умная батарейка».

Это потребовало бы значительно диверсифицировать производство батареек, но с такими преобразованиями в промышленности мы уже сталкивались. Сейчас день мы рассматриваем батареи как предмет потребления. Фактически батареи выпускаются самых разных типоразмеров, в зависимости от требований к их электрическим параметрам и от готовности покупателя истратить средства; например, перезаряжаемые батареи

¹ Конечно, можно включить батареи в комплект с устройством при покупке.

стоят дороже. Превращение такого товарного производства в более пестрое «рыночное» может показаться противоестественным. Но на самом деле это не так: достаточно вспомнить о бесконечном разнообразии покрышек (для легковых машин, грузовиков, тракторов, зимние, гоночные, высокопрочные, не говоря уже о головокружительном количестве размеров и форм-факторов), накопленных сегодня на складах для массового потребления. Тем не менее большинство людей считает покрышки товаром. Сто лет назад так оно и было.

Ясно, что система распространения батарей должна будет измениться: по-прежнему останутся сегодняшние «глупые батареи» наряду с «умными батареями», содержащими программное обеспечение. Не в каждом бакалейном магазинчике или ларьке сувениров найдутся батареи обоих типов. Однако можно рассчитывать, что с ростом спроса на умные батареи все больше магазинов станет ими торговать. Свободный рынок хорошо умеет решать такие проблемы.

Такие изменения придадут дополнительный смысл фразе «включить питание». Включая устройство, вы питаете его электрической энергией за счет батареи, но вы также снабдите его *интеллектуальной* энергией за счет ПО, находящегося на батарее.

Еще раз об обновлении ПО

В таком новом режиме, когда ПО покупается вместе с батареями, обновление происходит сразу. Когда батареи разряжаются, вы их заменяете. И получаете новейшую версию ПО для своего устройства.

Обратите внимание: срок службы батареи привязывается к сроку службы ПО, поэтому вопрос об устаревших версиях снимается с повестки дня.

А сколько будет стоить обновление или, в более общем случае, ПО? Никто, находясь в здравом уме, не станет требовать, чтобы батареи для его сотового телефона были бесплатными. Поэтому батареи станут, конечно, немало дороже, чтобы покрыть расходы на разработку ПО. Но эти расходы будут распределены на все батареи, нагруженные этим ПО.

И еще одно замечательное обстоятельство. Тот, кто мало пользуется своим устройством, редко будет менять батареи. Однако каждый раз при этом он будет обновлять свое ПО до новейшей версии. Напротив, тот, кто эксплуатирует устройство постоянно, будет часто менять батареи и обычно заменять при этом свое ПО точной его копией. Такой пользователь если и будет замечать изменения в работе программы, то редко, тогда как тот,

кто пользуется устройством мало, будет отмечать при смене батарей существенные изменения, но редко. Обновление ПО будет хорошо соответствовать схеме работы с устройством.

Поскольку ПО, поставляемое вместе с батареей, будет самодостаточным, будет навсегда покончено и с программными заплатками. Прежняя программа выбрасывается вместе со старой батареей, а новая батарея содержит свежую программу. Производители должны следить, чтобы изменение функций не оказывалось слишком резким. В крайнем случае новые батареи должны снабжаться сообщением об особенностях версии ПО. Думаю, что специалисты по маркетингу не преминут воспользоваться случаем превратить это в привлекательную особенность.

Большое преимущество такой модели заключается в том, что пользователь не должен загружать обновление ПО из Интернета. Что проще – загружать и устанавливать программу или просто заменить батарейки?

Некоторые приятные следствия

Устройства со встроенным ПО должны быть простыми в эксплуатации. Перефразируя покойного Джефа Раскина, «не существует группы пользователей Maytag, потому что в таковой нет нужды». Многие считают, что для таких устройств должна быть принята модель «бытовой техники».¹

К сожалению, иногда ПО мешает этому.

Но рассмотрим некоторые выгоды объединения батарейки и ПО. Надо установить программное обеспечение? Вставьте *чип-батарею*.² Это должно придать новый смысл термину «plug and play». Хотите перенести ПО с одного устройства на другое (совместимое)? Выньте чип-батарею из одного устройства и вставьте в другое. Если устройства несовместимы, это примерно то же, что пытаться вставить не ту батарею. Для большинства людей это не составит трудности.

Заметим, что для действия такой схемы в чипе-батарее должна быть какая-то доступная для записи память, чтобы хранить индивидуальную информацию (настройки и файлы), иначе в переносе чипа-батареи в другое устройство не будет особого смысла. Это может быть память типа PROM или NVRAM, а раз она интегрирована с батареей, то и обычная RAM.

¹ Я согласен с тем, что стиральная машина Maytag не является карманным устройством.

² Благодарю Кэйт Джонс за предложенный термин «batterychip».

Вот и еще одна причина обновить чип-батарейку – желание получить больше памяти.

Наши европейские друзья уже вступили на этот путь. Мой друг и коллега Паскаль Леруа пишет из Франции:

«Европейская телефонная система GSM основана на чипе, который называется SIM-картой и содержит 1) сведения о правах, подписке, номере телефона и т. д. и 2) ваши настройки. Переставление SIM-карты в другой телефон стало обычным делом: на днях я ехал в поезде с двумя девушками, у одной из которых был сотовый телефон с севшим аккумулятором. Она попросила телефон у своей подруги, вставила в него свою SIM-карту и болтала по телефону до конца поездки. Девушки не были похожи на технарей, так что применение такой технологии не должно вызвать затруднений практически ни у кого».

Этот пример «простоты эксплуатации» демонстрирует некоторые потенциальные выгоды чипов-батареек и позволяет развить идею еще дальше.

У вас несколько устройств, в которых зашито одно и то же ПО? Раньше такая ситуация тревожила производителей. Нередко можно было без труда (хоть формально это и незаконно) приобрести лицензионный экземпляр ПО и установить его на несколько машин. Применение чипа-батарейки делает вопрос обсуждаемым и для производителя и для пользователя. Можно купить несколько чипов-батареек и эксплуатировать их одновременно, а можно вставить чип-батарейку в то устройство, с которым вы сейчас хотите работать, а потом, если понадобится, переставить ее в другое. Как и в случае обычных батареек, вы делаете выбор между стоимостью и удобством. Это иллюстрация идеальной модели, в которой один экземпляр ПО соответствует одному физическому устройству. Если сделать эти чипы-батареи достаточно простыми и дешевыми, то нелегальное копирование ПО станет менее привлекательным.

На чем основана жизнеспособность такой схемы

На экономике. Людям не нравится платить абонентскую плату. Они не любят фиксированных текущих расходов. Когда экономическая ситуация ухудшается, люди сокращают расходы и прежде всего, если они умны, по незаметным ежемесячным счетам. Если производители ПО решат пойти по пути абонентской платы, то, как мне кажется, в хорошие времена их ждет процвета-

ние, а в плохие им придется тяжело. Здесь действуют элементарная экономика и психология.

Основная причина лежит в страхе, что деньги, вносимые в качестве абонентской платы, не окупят себя. Люди всегда беспокоятся, что их уровень пользования услугой будет ниже среднего, а потому окажется, что они финансируют каких-то любителей «бесплатной езды», которые косвенно получают услугу за чужой счет.

С другой стороны, платить за батарейки люди привыкли. Батарейки – это расходный материал, и *мы платим за них пропорционально израсходованной энергии*. Больше эксплуатируете устройством – быстрее сядут батарейки. На это никто не жалуется. Большинству это кажется честным.

Проиллюстрирую свою точку зрения на примере чернильных картриджей для дешевых цветных принтеров. Картриджи относительно дороги, но рынок это терпит, потому что они считаются расходным материалом. Но на вторичном рынке действует масса поставщиков, что доказывает применимость данной модели, т. к. их единственное назначение – снизить стоимость одного отпечатка.

Поэтому, интегрируя ПО с батареей, вы превращаете его в расходный материал, который выбрасывается вместе с разрядившейся батареей. Стоимость ПО должна войти в цену батареи. Благодаря разделению стоимости на очень много частей увеличение цены должно оказаться очень незначительным. Привязка ПО к батарее может оказаться простейшим алгоритмом взимания платы за него, основанным на модели его применения.

Станут ли люди перезаряжать свои разрядившиеся батареи с ПО? Могут, если мы будем устанавливать ПО на батареи, которые допускают перезарядку. В таком случае они смогут хранить свое старое ПО чуть ли не вечно. Но принципиальная модель от этого не пострадает. И при желании всегда можно обновить ПО, выкинув «хорошую» батарейку и установив новую. Как знать, возможно, возникнет вторичный рынок «старых» батареек, которые сохранили заряд. Может быть, станут практиковать обмен батареек. Я мог бы предложить массу изменений для базовой модели. Важно то, что свободный рынок и его экономика поставят все на свои места.

Уточнение

Можно рассмотреть еще некоторые детали, которые могут понравиться любителям технологических новшеств. Объединение ПО с батареями открывает ряд новых горизонтов.

Одно уточнение, предложенное Филиппом Крухтенем, состоит в том, чтобы с помощью памяти большого объема хранить ПО для нескольких устройств на одной батарее. Это несколько облегчило бы проблему пространства, поскольку один физический блок годился бы для нескольких различных устройств. Можно представить себе универсальную батарейку «для сотовых телефонов» и т. д. В такой ситуации можно было бы заменить телефон Motorola на Nokia, вставить старую батарейку, и все заработало бы как надо. Конечно, для этого производители сотовых телефонов должны договориться о каких-то стандартах, а это всегда бывает трудно.

Можно, наоборот, представить себе компанию с вертикальной интеграцией, которая поместит в одну чип-батарейку ПО для устройств нескольких разных типов. Тогда вы, скажем, купите умную батарейку Ericsson, и разные устройства, выпускаемые Ericsson, будут работать от ее энергии и программ.

Сегодня существует множество энергосберегающих программных алгоритмов для экономии заряда батареи. Мы уже пользуемся ими в ноутбуках, которые поставляются вместе с программным обеспечением. Можно поместить эти энергосберегающие алгоритмы туда, где их присутствие более логично, – в аккумулятор ноутбука.

Возьмем также батарейки поколения «n» с программным обеспечением поколения «n». Допустим, что в следующей версии ПО алгоритмы и прочее усовершенствуются, и для выполнения той же работы за то же время потребуется меньше электрической энергии. Это означает, что ПО поколения «n+1» можно разместить на батарее меньшей электрической мощности без ущерба для конечного результата. В результате может снизиться цена батареи, или вырасти ваша прибыль, или то и другое вместе.

Все эти возможности выглядят реальными в рамках современных технологий, но это не столь важно. Это лишь развитие изначальной идеи, а я хочу по возможности придерживаться принципа KISS.¹

Как быть с программным пиратством?

В принципе я хочу ликвидировать пиратство, сделав чип-батарейки экономически самым привлекательным вариантом для большинства пользователей. Как и во всех других системах, пытаясь создать технически более изощренную мышеловку, мы только поощряем самых умных мышей.

¹ KISS = Keep It Simple, Stupid! (будь проще...)

Лучше придумать, как вызвать у них желание покупать сыр. Возможно, память умных батареек будет программироваться в заводских условиях с помощью относительно дорогих устройств, не доступных широкой публике. Тогда дешевле будет купить новую чип-батарею вместо того, чтобы копировать ПО и пытаться «прожечь» его в старой батарее. Но в принципе я не ставил себе задачи борьбы с пиратством. Возможно, какие-то благотворные побочные эффекты это предложение повлечет, но они случайны.

Пока солнце не взяло верх

Сегодня тратится много средств на то, чтобы сделать батарейки лучше (меньше, мощнее, долговечнее и т. д.), чтобы сделать нас более мобильными. Возможно, когда-то появятся заряжаемые от солнца конденсаторы, которые совершенно вытеснят батарейки. Для успеха этой технологии необходимо обеспечить достаточно маленький форм-фактор, достаточно большую емкость и совершенный интерфейс с солнечной панелью. Конденсатор станет применяться в качестве виртуальной батареи. Это будет просто еще один способ запасти энергию впрок. Однако, в отличие от батареек, комплект конденсатор/солнечная панель не будет требовать периодической замены. Пока не наступил этот день, мы будем менять или перезаряжать батареи. Так почему бы не объединить их с нашим ПО?

Было бы безрассудством с моей стороны претендовать на открытие очередного великого вертикального объединения нашей эпохи – батареек с программами. Но эта идея меня заинтриговала. Мне нравится исследовать плюсы и минусы таких схем, без каких-либо конечных результатов.

Чипы-батареи решили бы проблему обновления ПО электронных устройств, упростив эту операцию до смены батареек. Они также изменили бы динамику ценообразования, сделав ПО скорее расходным материалом, чем капитальным товаром. Когда устройства со встроенным ПО сами станут товаром, это будет оправданно. В итоге могут возникнуть интересные проблемы распространения чипов-батареек, но я убежден, что их можно эффективно решать.

Резюме

За несколько лет, прошедших после первоначального моего предложения, взрыва энтузиазма по поводу чип-батареек не произошло. Но иногда для таких идей требуется время. Самое большое возражение исходит

от тех, кто пользуется перезаряжаемыми батареями; они заявляют, что поскольку не меняют свои аккумуляторы, то и новое программное обеспечение никогда не получают, и что в результате вырастет цена одного экземпляра программного обеспечения, поставляемого с батареей.

Я не уверен в том, что это действительно проблема. Во-первых, сегодня много как тех, кто пользуется разовыми батарейками, так и тех, кто пользуется перезаряжаемыми. Для большинства из них это выбор между ценой и удобством, и рынок так корректирует цены, что место находится для всех. Аналогично, как мне кажется, цена программного обеспечения, интегрированного с каждым типом батарей, будет определяться сроком службы батареи. Поэтому одно и то же программное обеспечение будет дороже при установке его в перезаряжаемую батарею, чем в одноразовую, потому что срок службы его будет дольше. Это вопрос ценообразования, и рынок его решит.

Главная идея, с моей точки зрения, – это перевод программного обеспечения из категории капитальных затрат в категорию расходных материалов. В результате логичным становится ценообразование в зависимости от системы пользования, а это в конечном счете должно быть полезнее всего для рынка.

До встречи в магазине умных батареек!

ГЛАВА ДВАДЦАТАЯ

Числа случайные и не очень

В современную эпоху тот, кому необходимо выполнить расчеты, располагает невообразимо богатым арсеналом. К сожалению, мы часто забываем о самом мощном из них – о своем мозге. Я хочу этим сказать, что иногда при необходимости что-то выяснить мы бросаемся открывать новую электронную таблицу вместо того, чтобы сначала подумать, что собственно мы хотим. Инструмент не должен определять метод решения. Как тут не вспомнить старую поговорку: когда у вас нет ничего, кроме молотка, все кажется похожим на гвоздь.

От этой напасти есть хорошее средство, к которому я обращаюсь уже многие годы. Я представляю себе, что нахожусь на необитаемом острове и должен решить некую задачу. Как бы я стал искать решение, пользуясь только инструментами Архимеда – своей головой и палочкой, чтобы чертить на песке? (Конечно, хорошо было бы иметь голову Архимеда, но выбирать не приходится). Так что правила простые: разрешается делать любые вычисления, которые можно написать на песке.¹ Какие задачи все же удастся решить?

Так вот, оказывается, что можно делать довольно интересные вещи. Удивительно, что некоторые задачи, которые вначале выглядят совсем простыми, требуют больше размышлений и вычислений, чем кажется на первый взгляд. Далее в этой главе рассказывается о Роско Леруа и одной

¹ Песка сколько угодно, как будто у вас неограниченный запас бумаги. В основном это означает, что необязательно запоминать все промежуточные результаты: их можно «сохранить» на будущее.

такой задаче.¹ Я все же разрешу ему пользоваться огрызками карандашей, бумагой и старой доброй логарифмической линейкой, но, как я отмечу далее, это все-таки предметы роскоши.

Роско описывает ситуацию

– Помнишь, как я однажды потерпел крушение в южной части Тихого океана вместе с моим приятелем, которого звали Понедельником? – начал свой рассказ Роско. Имя Понедельник, надо сказать, Роско придумал в честь Пятницы Робинзона Крузо, и этот Понедельник был его спутником в путешествиях и помощником во всех делах. Роско и Понедельник не одну ночь провели под открытым небом, и если у кого был шанс остаться в живых на необитаемом острове, так это у них.

– Да, – отвечал я. – Вам, ребята, повезло: никто, кроме вас, не уцелел, насколько я помню. Это был тропический остров, еды вдоволь и укрытие из обломков. Оставалось только ждать, когда вас спасут.

– Да, – согласился Роско, – нам *оставалось* только ждать. Ускорить события не было никакой возможности. Главной проблемой стала скука: в обломках кораблекрушения не оказалось никаких книг, и нам крайне необходимо было найти какое-то времяпрепровождение, чтобы не сойти с ума.

Оказалось, что при осмотре выброшенного на берег имущества Роско и Понедельник обнаружили целую кучу бейсбольных карточек. Спустя какое-то время Понедельник, заядлый бейсбольный болельщик, предложил воспользоваться статистикой на бейсбольных карточках и, чтобы коротать время, создать две команды и проводить между ними воображаемые игры. К счастью, карандашей и бумаги у них было вдоволь. У Роско даже сохранилась карманная логарифмическая линейка.

– Как только Понедельник предложил эту идею, – сказал Роско, – я сразу увлекся ею. У нас появилось бы занятие и невинная форма соперничества. Поэтому я стал соображать, как построить игру. Тут и возникла эта трудная задача.

Моделирование отбивающего

– Для всякого моделирования такого рода нужен какой-то генератор случайных чисел, – продолжал Роско.

¹ Мы уже встречались с Роско в главах 5, 10, 11, 14 и 15.

– Так, – сказал я, – и какие же у вас оказались для этого приспособления?

– Не так много, – отвечал Роско. – У нас было три одинаковых обычных игральные кости, от одной до шести точек на каждой грани. И я решил, что с их помощью легко будет смоделировать вероятности, потому что именно они нам были нужны. Мы решили, что проще всего бросать все три кости одновременно, складывать очки и по сумме определять успех или поражение.

– Вероятности? Я не совсем понимаю, – сказал я.

– Разумеется, вероятности! В этом и есть идея воображаемого бейсбола. Допустим, что у базы стоит отбивающий, средний результат которого равен 0,250. На первом уровне приближения (забудем на минуту о прогулках) мы случайным образом определяем, удалось ли ему отбить мяч, для чего нам надо получить случайное событие с вероятностью 250 из 1000. – Никогда не замечал у Роско такого интереса к теории вероятностей и статистике, но меня еще многое ждало впереди.

– В действительности все сложнее. Например, в нашей игре мы не обращали внимания на мастерство питчера и более сложные ситуации типа жертвенных мячей и т. д.

– Хорошо, допустим, вы сделали все эти упрощения. Как вы моделируете действия отбивающего в доме? – спросил я.

– Мы должны позаботиться только о двух вещах, – отвечал Роско. – Выступлениях в доме (plate appearances – PA) и официальных выходах на бит (at-bat – AB). Если игрок получает «базу на болах», или «прогулку», это засчитывается ему как PA, но не как официальный AB. Поэтому в принципе нужны два числа: процент от выступлений в доме, за которые игрок получает AB, и его показатель отбивания (batting average). Допустим, например, что игрок получает базу на болах один раз из каждых 10 выступлений в доме.¹ Тогда надо сначала определить, получает ли игрок прогулку, путем случайного испытания с вероятностью успеха 0.1. Если есть три кости, необходимо выяснить, какое количество очков выпадает с такой вероятностью. Так что бросаешь кости, получаешь эту сумму – и вперед! Отбивающий идет на первую базу и все в порядке. С другой стороны, если выпадет другая сумма, игрок не получает «прогулку». Вместо этого у него официаль-

¹ Отметим, что такая статистика, как выходы на бит и базы по болам (из которых получается количество выступлений в доме), есть на бейсбольных карточках, которые нашлись среди обломков.

ный выход на биту. Допустим, что средний показатель отбивания у него .250 и есть событие на костях, у которого такая вероятность. Если оно выпало, то он успешно отбил мяч, если нет, значит, он попал в аут. Можно, конечно, по статистике выходов на биту моделировать тип удара – одиночный, двойной, тройной или хоумран. И так далее. Я взял в качестве примера средний показатель отбивания, но можно детализировать модель по своему вкусу в зависимости от того, сколько «сложных ситуаций» ты хочешь учесть.¹ Однако все равно все сводится к моделированию события с определенной вероятностью. А т.к. иногда нам нужны вероятности для других вещей, а не только для показателя отбивания, желательно иметь возможность охватить весь диапазон – от нуля (полная неудача) до 1 (достойный успех).

– Достаточно понятно. Суть ясна. Так в чем проблема? – поинтересовался я.

– Проблема в следующем, – ответил Роско. – Можно ли моделировать вероятности с помощью такого простого приспособления, как три игральные кости, и при этом обеспечить необходимое разрешение? Например, если мы сможем моделировать только вероятности .250, .500 и .750, то этого будет недостаточно для хорошей модели.

– Начать с того, что если ты станешь бросать три кости одновременно, то получишь суммы от 3 до 18, т.е. 16 разных исходов, – отвечал я.

– Верно, – сказал Роско, – именно так мы и приступили к делу.

Первые шаги

– Понедельник разобрался с комбинаторикой. Для трех костей с шестью гранями возможны $6 \times 6 \times 6$ исходов, или 216 возможностей, но всего 16 разных сумм. Итак, у него получилась табл. 20.1.

¹ Некоторые зарабатывают себе этим на жизнь, строя разного рода вымышленные модели; это естественно, поскольку знатоки бейсбола регистрируют любую мыслимую статистику. Надо сказать также, что наша схема моделирования игры отбивающего не единственная возможная. Например, можно заново рассчитать средний показатель отбивания по выступлениям в доме, а потом рассчитать прогулки, синглы, даблы, трайплы и хоумраны по ренормализованной статистике. Существует масса альтернативных формулировок, из которых мы выбрали лишь одну. Однако в любом случае до широкого распространения компьютеров возникала проблема моделирования вероятностей с помощью дешевых устройств, доступных большинству людей.

Таблица 20.1. Количество вариантов для получения различных сумм

Сумма	Число способов ее получить
3	1
4	3
5	6
6	10
7	15
8	21
9	25
10	27
11	27
12	25
13	21
14	15
15	10
16	6
17	3
18	1
Сумма	216

– Это мне нравится, – отвечал я. – Симметрия видна невооруженным глазом. Например, 3 и 18 появляются по одному разу, как и должно быть. Для 4 и 17 значения одинаковы, и т. д. Чаще всего получаются 10 и 11, потому что эти суммы дают многие комбинации. И в сумме получается 216, так что все, возможно, правильно. Но, по всей видимости, получится всего восемь разных значений для вероятностей.

– Видимость бывает обманчива, – отвечал Роско. – Но для большей надежности сложим-ка те вероятности, которые у нас уже есть. – И он показал табл. 20.2.¹

¹ Обратите внимание, что в вероятностях встречается по пять значащих цифр. Роско на своем острове мог с помощью линейки получить не больше трех. В числе 0,04630 три значащих цифры, остальные следует округлить. Хотя вероятность, как частное двух целых, имеет неограниченную точность, Роско с логарифмической линейкой мог получить только три цифры. Но если бы он выполнил деление вручную, то получил бы столько цифр, на сколько у него хватило бы времени и терпения.

Таблица 20.2. Вероятности, соответствующие каждой сумме

Сумма N	Число способов ее получить	Вероятность N
3	1	0,00463
4	3	0,01389
5	6	0,02778
6	10	0,04630
7	15	0,06944
8	21	0,09722
9	25	0,11574
10	27	0,12500
11	27	0,12500
12	25	0,11574
13	21	0,09722
14	15	0,06944
15	10	0,04630
16	6	0,02778
17	3	0,01389
18	1	0,00463
Сумма	216	1,00000

– Надо же! – воскликнул я. – Роско умеет делить на 216!

– Спокойно, сынок, – сказал Роско. – Все только начинается.

Следующие шаги

– Должно быть очевидно, что если ты хочешь смоделировать вероятность 0,00463, то для успеха банкомет должен выбросить 3 очка. Конечно, можно ждать и 18 – симметричного результата, но нам нужны отдельные вероятности, поэтому мы выбираем что-нибудь одно. Выберем 3. – Роско подождал секунду и выдал свой первый сюрприз.

– Но что будет, если определить «успех» как выпадение 3 или 4 очков? Вероятность выбросить 3 составляет 0,00463, а вероятность выбросить 4 составляет 0,01389, поэтому вероятность выбросить 3 или 4 составляет просто 0,00463 + 0,01389, или 0,01852. Теперь ты видишь, что можно соз-

дать некоторые другие отличные значения вероятности, рассматривая как «успешные» несколько сумм.

– Так, правильно ли я понял? – отвечал я. – Чтобы моделировать 0,00463, мне надо, чтобы выпала тройка. Чтобы моделировать 0,01389, мне надо, чтобы выпало число 4. А чтобы моделировать 0,01852, мне нужно, чтобы выпало 3 или 4. Только и всего?

– Да, ты понял правильно. Но как ты определишь, какие еще есть возможности? – Улыбка на лице Роско граничила с ухмылкой. Я всерьез задумался.

Получение дополнительных вероятностей

– Так, – сказал я, – добавляем в твою таблицу еще колонку. Какие-то новые вероятности можно получать, рассматривая другие «совокупные» исходы. – Я поправил таблицу Роско, и получилась табл. 20.3.

Таблица 20.3. Дополненная таблица вероятностей

Сумма N	Число способов выкинуть N	Вероятность N	Вероятность N или меньше
3	1	0,00463	0,00463
4	3	0,01389	0,01852
5	6	0,02778	0,04630
6	10	0,04630	0,09259
7	15	0,06944	0,16204
8	21	0,09722	0,25926
9	25	0,11574	0,37500
10	27	0,12500	0,50000
11	27	0,12500	0,62500
12	25	0,11574	0,74074
13	21	0,09722	0,83796
14	15	0,06944	0,90741
15	10	0,04630	0,95370
16	6	0,02778	0,98148
17	3	0,01389	0,99537
18	1	0,00463	1,00000
Сумма	216	1,00000	

– Что ж, ты на правильном пути, – реагировал Роско. – В твоей четвертой колонке явно появились новые вероятности. Например, вероятность выпадения 3 или 4 равна $0,00463 + 0,01389$, или, как ты вычислил, 0,01852. Вместо «3 или 4» можно сказать «4 или меньше». Говоря «5 или меньше», ты имеешь в виду, что успешный исход составляет выпадение суммы 3, 4 или 5. На самом деле мы с Понедельником тоже действовали именно в этом направлении. Только это далеко не все.

– Прежде чем ты двинешься дальше, – сказал я, – давай посмотрим, сколько вероятностей у нас уже есть. Я отмечу в таблице все различные вероятности. – И я добавил затенение, получив табл. 20.4.

Таблица 20.4. Дополненная таблица вероятностей с затенением

Сумма N	Число способов выкинуть N	Вероятность N	Вероятность N или менее
3	1	0,00463	0,00463
4	3	0,01389	0,01852
5	6	0,02778	0,04630
6	10	0,04630	0,09259
7	15	0,06944	0,16204
8	21	0,09722	0,25926
9	25	0,11574	0,37500
10	27	0,12500	0,50000
11	27	0,12500	0,62500
12	25	0,11574	0,74074
13	21	0,09722	0,83796
14	15	0,06944	0,90741
15	10	0,04630	0,95370
16	6	0,02778	0,98148
17	3	0,01389	0,99537
18	1	0,00463	1,00000
Сумма	216	1,00000	

– Я насчитал 8 в одной колонке и 14 в другой, т. е. всего 22. Это правильно? – спросил я.

– Не совсем, – отвечал Роско. – У нас наблюдается, так сказать, *случайное вырождение*. Вероятность выпадения 6 совпадает с вероятностью вы-

падения «5 или менее». Это потому, что 10 способов дают 6, и $(1 + 3 + 6 = 10)$ дают 3, или 4, или 5, т. е. 5 или меньше. Так что я думаю, у нас пока 21 раз-личная вероятность. Но, как я сказал, это далеко не все. Есть множество других комбинаций.

Про бейсбол мы, конечно, уже забыли

Задача несколько осложнялась. – И что же вы предприняли дальше? – спросил я у Роско.

– Ну, кое-что стало понятно, – отвечал Роско. – Понедельник усадил меня и стал рассказывать умные вещи. Для начала он сообщил, что если мы хотим генерировать вероятности для бейсбола, то выбрали не лучший путь. Он предложил систему, в которой можно было бы бросать одну кость несколько раз и генерировать все то, что нам надо для бейсбола. Я вынужден был с ним согласиться.

Роско продолжил. – Он также сказал, что задача создания вероятностей по сумме трех одинаковых костей, которые бросают одновременно, интересна сама по себе. Она теперь больше интересовала его, чем изначальная проблема. Поэтому мы решили, что попытаемся рассмотреть ее в самом общем виде. Кстати, так иногда бывает в жизни, – заметил Роско. – Начиная решать одну задачу и обнаруживаешь по ходу дела другую, более интересную. Кажется, это называется талантом делать случайные открытия или как-то в этом роде.

Действительность уродлива

– Затем мы решили, что определение всех возможных комбинаций становится слишком трудоемким. Мы начали составлять таблицы, но быстро отказались от этого. Понедельник сказал, что надо на время отложить это.

Роско закурил сигару, и я решил, что сейчас последует завершение истории. – Надо сказать, Понедельник предложил решение задачи уже на следующий день, – продолжал Роско.¹ – Он пришел к выводу, что прежде всего

¹ Кое-кто из читателей может придти к выводу, что Роско олицетворяет вашего покорного слугу. Если это так, то Понедельник – это воплощение моего сына Дэвида, который нашел решение этой проблемы. Возможно, мне неплохо удастся открывать новые проблемы, но еще лучше Дэвиду удастся их решать, а кроме того, он ярый бейсбольный болельщик. А имя Понедельник дано этому воплощению потому, что в этот день мы приходим на работу.

надо определить верхнюю границу числа возможных вероятностей. Поскольку есть всего 216 способов, которыми могут выпасть кости, это и есть максимум. Поэтому в лучшем случае мы могли разбить интервал от нуля до 1 на 216 шагов. С точки зрения величины разрешения оптимальное решение должно иметь постоянный шаг в $1/216$, или 0,00463.

– Как генерировать первую вероятность, мы уже знаем! – сказал я.

Роско усмехнулся.

– А почему ты считаешь, что важно определить максимально возможное количество вероятностей? – спросил я.

– Потому что, – сказал Роско, – если этого не сделать, возникнет проблема. Допустим, что ты покопаешься и найдешь 87 различных вероятностей, а не 21, как мы пока смогли.¹ Как узнать, что ты нашел их все и ничего не пропустил? Зная максимальное число, ты можешь остановиться, когда достигнешь его. Если нет, то придется выяснять, почему не получается найти остальные. На самом деле надо выяснить, сможем ли мы реализовать 108 возможностей до вероятности 0,5. Остальные 108 получатся как дополняющие решения.

Это было разумно. В данной области стандартно применяется такой прием. Если ты знаешь, что 3 выпадает с вероятностью 0,00463, то выпадение чего угодно, кроме 3, имеет вероятность $(1 - 0,00463)$, или 0,99537. Поэтому получить вероятности до 0,5 всегда бывает достаточно.

Решение Понедельника

– Понедельник проявил систематический подход. В первую очередь он занялся количеством способов, которыми может быть получена сумма, зная, что всегда может преобразовать эти числа в вероятности, поделив на 216. Обращаться с целыми числами легче, чем с десятичными дробями.

– Сначала он решил выяснить, – продолжал Роско, – все ли начальные девять «путей» могут быть построены. У него получилась табл. 20.5.

Таблица 20.5. Как получить первые 9 способов

Способов	Нужно выбросить сумму
1	3
2	3 или 18

¹ Как это сделал сам Роско в одной из первых попыток решить эту задачу.

Способов	Нужно выбросить сумму
3	4
4	3 или 4
5	3, или 4, или 18
6	5
7	3 или 5
8	3, или 5, или 18
9	4 или 5

– Кажется, понимаю, – сказал я. – Понедельник пытается выяснить, сможет ли он построить полный набор «способов» вплоть до 108 из элементов колонки «количество способов» – 1, 3, 6, 10, 15, 21, 25 и 27. Очень толково.

– Да, – ответил Роско, – мысль именно такая. Но помни, что каждым элементом он может воспользоваться только дважды. Симметрия помогает, но обрати внимание, что «способы» расходуются, поэтому необходима внимательность. Например, предположим, что мы уже использовали 3 и 18 и нам нужен еще один «путь». У нас ничего не выйдет. Поэтому придется соблюдать известные ограничения.

– А-а, – сказал я, – так ответ все еще под сомнением.

– Понедельник оказался на высоте, – сказал Роско. – Вот как он рассуждал дальше. Получается, что есть 10 способов выпадения 6. Если взять 6 и дополнительное к нему 15, получатся две десятки, стало быть, мы можем теперь конструировать количество способов от 1 до 29. Теперь мы можем везде добавлять от 1 до 29, если при этом нам не понадобятся дополнительно суммы 3, 4, 5, 6, 15, 16, 17 и 18. Мы израсходовали все элементы количества способов 1, 3, 6 и 10.

– Ну, от 29 до 108 все еще далековато, – сказал я.

Роско изложил решение Понедельника до конца. – Если теперь использовать одну из сумм с 21 путями, например 8, то диапазон расширится с 29 до 50. А у тебя еще остаются по две суммы для 15, 25 и 27 путей. Взяв вторую сумму с 21 способом, расширим диапазон с 50 до 71. Пара по 27 способов дает нам 125 вариантов, а это больше нужных нам 108. Так что задача решается.

– Стало быть, ты утверждаешь, – отвечал я, – что реализуемы все варианты, т. е. мы покроем весь интервал с шагом в $1/216$. Это поразительно. Ты построил фактическую таблицу?

– Это было нетрудно, когда мы поняли, что это можно сделать, – сказал Роско. Окончательный результат приведен в табл. 20.6.

Таблица 20.6. Решение задачи

Способы	Вероятность	Суммы, дающие эту вероятность						
1	0,00463	3						
2	0,00926	3	18					
3	0,01389	4						
4	0,01852	3	4					
5	0,02315	3	4	18				
6	0,02778	5						
7	0,03241	3	5					
8	0,03704	3	5	18				
9	0,04167	4	5					
10	0,04630	6						
11	0,05093	3	6					
12	0,05556	3	6	18				
13	0,06019	4	6					
14	0,06481	3	4	6				
15	0,06944	7						
16	0,07407	3	7					
17	0,07870	3	7	18				
18	0,08333	4	7					
19	0,08796	3	4	7				
20	0,09259	6	15					
21	0,09722	8						
22	0,10185	3	8					
23	0,10648	3	8	18				
24	0,11111	4	8					
25	0,11574	9						
26	0,12037	3	9					
27	0,12500	10						
28	0,12963	3	10					
29	0,13426	3	10	18				

Способы	Вероятность	Суммы, дающие эту вероятность						
30	0,13889	4	10					
31	0,14352	6	8					
32	0,14815	3	6	8				
33	0,15278	3	6	8	18			
34	0,15741	4	6	8				
35	0,16204	6	9					
36	0,16667	3	6	9				
37	0,17130	3	6	9	18			
38	0,17593	3	6	10				
39	0,18056	3	6	10	18			
40	0,18519	7	9					
41	0,18981	3	7	9				
42	0,19444	8	13					
43	0,19907	3	8	13				
44	0,20370	3	8	13	18			
45	0,20833	6	9	15				
46	0,21296	8	9					
47	0,21759	3	8	9				
48	0,22222	8	10					
49	0,22685	3	8	10				
50	0,23148	9	12					
51	0,23611	3	9	12				
52	0,24074	9	10					
53	0,24537	3	9	10				
54	0,25000	10	11					
55	0,25463	3	10	11				
56	0,25926	3	10	11	18			
57	0,26389	4	10	11				
58	0,26852	6	8	10				
59	0,27315	3	6	8	10			
60	0,27778	5	10	11				
61	0,28241	3	5	10	11			
62	0,28704	6	9	10				

Таблица 20.6 (продолжение)

Способы	Вероятность	Суммы, дающие эту вероятность						
63	0,29167	3	6	9	10			
64	0,29630	6	10	11				
65	0,30093	3	6	10	11			
66	0,30556	3	7	9	12			
67	0,31019	7	9	10				
68	0,31481	3	7	9	10			
69	0,31944	7	10	11				
70	0,32407	3	7	10	11			
71	0,32870	8	9	12				
72	0,33333	3	8	9	12			
73	0,33796	3	8	9	12	18		
74	0,34259	3	8	9	10			
75	0,34722	8	10	11				
76	0,35185	3	8	10	11			
77	0,35648	9	10	12				
78	0,36111	3	9	10	12			
79	0,36574	9	10	11				
80	0,37037	7	9	12	14			

– Обрати внимание, – сказал Роско, – что некоторые решения не единственны. Но этого и не требуется. Нам достаточно найти лишь один набор сумм, который дает требуемое количество способов «попасть в него».

Полученные уроки

Когда Роско закончил рассказ, он не выглядел совершенно удовлетворенным. «Что тебя беспокоит, Роско?» – спросил я.

– Ну, во-первых, я опять обманулся, – отвечал он. – Я думал, что это задача для детей, а оказалось, что она гораздо сложнее. Это всегда раздражает.

Во-вторых, один из моих проверенных временем приемов подвел меня, – продолжал он. – Обычно, если задача не решается с ходу, я пытаюсь разобраться с ее упрощенным вариантом. Но в этом случае более простой случай суммы очков двух костей был совершенно бесполезен.

В-третьих, решение Понедельника убедительно, особенно когда держишь в руках итоговую таблицу. Но даже здесь возникает ощущение использования какой-то эвристики, а не логического доказательства. Хотя если кто-то решил задачу, я меньше всего склонен критиковать его за тот способ, которым он это сделал. Нельзя сказать, что я ратую за особую чистоту, а тем более элегантность.

Я бы отнесся к этому более милосердно. Роско, а особенно его приятель Понедельник, проделали огромную работу с помощью карандаша, бумаги, логарифмической линейки и здравого смысла. На самом деле, если бы Роско потерял свою линейку при кораблекрушении, Понедельник все равно смог бы решить задачу, потому что деление на 216 – это уже украшательство, и можно записывать вероятности в виде $67/216$.¹ Замечательно то, что можно решить эту задачу на необитаемом острове и без всяких компьютеров, Microsoft Excel и сводных таблиц, потратив только время, энергию и проявив интеллектуальное любопытство. Это также означает, что Блез Паскаль мог решить эту задачу в XVII веке: у него были все необходимые для этого инструменты. В недостатке интеллекта и любознательности его тоже не упрекнешь. Любопытно, что, как отмечает Хемминг, примерно в 1642 г. Галилей решал вопрос о том, какая сумма более вероятна при бросании трех костей – 9 или 10. Он добрался как минимум до нашей табл. 20.1, правильно рассудив, что 10 немного более вероятно, чем 9. Так что люди размышляли о подобных вещах задолго до появления бейсбола.

Роско и Понедельник в конечном итоге смогли моделировать равномерное распределение с точностью около *половины процента*,² просто бросая три кости и считая сумму очков, задав сначала вероятность, а потом решая, успешным было бросание костей или нет. Для исходного примера с бейсболом это означало возможность моделировать средний показатель отбивания с точностью до пяти пунктов.³

¹ Роско мог бы при необходимости выполнить деление вручную.

² Одна часть из 216 немного лучше, чем полпроцента, ровно полпроцента дала бы одна часть на 200.

³ Покойный Тед Уильямс в 1941 году получил .406 и был последним, кто смог превысить .400 в течение сезона. Следовательно, табл. 20.6 более подходит для показателя отбивания. Если вероятности больше .500, нужно симметричным образом расширить таблицу, используя упомянутый ранее принцип дополнителности.

Резюме

– И последняя забавная вещь, – сказал Роско в завершение. – Понедельник продемонстрировал полное понимание задачи, сделав такое наблюдение. Если тот же эксперимент проводить с четырьмя костями, то окажется, что *невозможно* равномерно охватить интервал от нуля до единицы. То есть вероятностей можно получить гораздо больше, но они не распределяются равномерно. *Это* интересный результат, справедливый для любого количества костей, большего трех. Ничего не поделаешь!

Рекомендую читателям проверить, смогут ли они сами, как Понедельник, доказать невозможность для четырех и более костей.

Вот примерно настолько я позволил себе в этой главе уклониться в сторону «нестандартного мышления». Речь здесь шла не столько о разработке ПО, сколько о том, чтобы не зашоривать свой ум и искать разные подходы к необычным задачам, с которыми мы постоянно сталкиваемся.

Перехожу к последней части книги, которую я назвал «Более сложные вопросы».

ЧАСТЬ ШЕСТАЯ



Более сложные вопросы

Мы завершаем последний круг и выходим на финишную прямую. В этой части я обращаюсь к темам, которые потребуются вам для оттачивания своего мастерства менеджера разработки ПО. Например, каждый хороший менеджер рано или поздно получает предложение возглавить незаконченный проект, успешное завершение которого оказалось под угрозой.

Глава 21 «Кризис» иносказательно рассматривает ситуацию, когда вы попадаете в самую гущу событий и пытаетесь найти правильную дорогу.

В главе 22 «Расширение» рассматривается случай, когда вам предлагают увеличить свою организацию. Возможно, вы только что успешно завершили проект и ваш босс хочет поручить вам новый, более крупный. Можно ли увеличить численность команды и не сделать ее при этом менее эффективной?

Когда вы приступаете к управлению чем-то более крупным, чем один проект, то задумываетесь над тем, какого рода организацию вы хотите построить. В главе 23 «Культура» рассмотрены вопросы культуры и ценностей – тех неуловимых сил, которые правят где-то высоко и создают атмосферу, в которой должны действовать все команды.

Наконец, глава 24 «Соединяем все вместе» завершает книгу философскими рассуждениями о проявлениях человеческой природы при карьерном продвижении.

ГЛАВА ДВАДЦАТЬ ПЕРВАЯ

Кризис

Каждого удачливого менеджера «награждают» тем, что предлагают все более и более трудную работу. Законы профессионального роста заставляют браться за новые и разные проблемы. Одна из самых трудных среди них – это принятие на себя руководства чужим проектом в середине его выполнения.

Обычно руководство очень неохотно идет на смену менеджера в середине программного проекта. Менеджер, как правило, хранит у себя в голове столько «контекста», что его замена связана с большими издержками. Но иногда другого выбора не остается. Замена неизбежна, выбор пал на вас, и вы должны принять проект и навести в нем порядок.

Во-первых, вы должны понимать, что у вас на руках оказался больной пациент. Если бы ситуация не была столь угрожающей, никто не стал бы менять доктора, поэтому не рассчитывайте, что вы придете туда и все будет чудесно. Во-вторых, когда проект находится в таком состоянии, многое, видимо, было сделано неправильно. В-третьих, готовьтесь к тому, что ситуация будет еще хуже, чем кажется вашему начальнику: обычно предстоит узнать еще другие скверные новости. Разгребая завалы, вы наверняка найдете всякие мерзости, которые заставят вас содрогнуться.

Но зато вам не придется осторожничать. Когда человек тяжело болен, он не станет жаловаться, что лекарство горькое. Следует действовать быстро, принимать решения и двигаться вперед. Не бойтесь все изменить. Кризис не обходится без своего вечного спутника – цейтнота. Но даже если

времени достаточно, путь к спасению редко лежит в прежнем направлении. Надо менять курс и делать это быстро.

Но что именно делать, и в каком порядке?

Пять дней рыбы

Дохлая рыба начинает пахнуть на второй день, а на третий она уже издает зловоние. Бизнес-проблемы похожи надохлую рыбу. Обычно люди замечают их, только когда появляется дурной запах, т. е. процесс разложения уже какое-то время шел. Если вас просят что-то сделать в ситуации, когда идет разложение, то очень важен фактор времени. Вот некоторые рекомендации, облеченные в аллегорическую форму и призванные помочь вам проанализировать и разрешить кризис.

Рыбный рынок

Представим себе некий экзотический рыбный рынок, где рыбу держат в аквариумах, чтобы вы могли купить ее и съесть свежей. В таких условиях цены будут высоки, издержки велики, объем торговли мал и прибыль невысока.

При этом рынке состоит на службе небольшая, но очень шумная группа кассандр, которые периодически возвещают, что рыба вот-вот сдохнет, хотя рыбки плавают в аквариуме как ни в чем не бывало. Вот состав этой группы:

- Работники, которые недостаточно загружены работой и постоянно ищут, что не в порядке. И их никогда не привлекают к ответственности, если их предсказания не оправдываются. Поэтому они со всей энергией предсказывают всякие напасти, изредка попадая в точку (согласно законам статистики). Соотношение сигнал/шум у них очень невысоко, поэтому хозяин обычно может спокойно не обращать на них внимания.
- Несколько человек, которые действительно *в состоянии видеть*, что происходит, и не боятся говорить. Если хозяин в состоянии отличить этих людей от первой группы, они могут быть ему полезны, но и они не всегда оказываются правы.

День 1: Неведение

И вот одна рыба погибает, но мало кто замечает это. Пара бестолковых работников глазееет на нее, обсуждая между собой, нормально ли это, что рыба всплыла кверху брюхом. Потом они будут говорить, что заметили что-то подозрительное, но не были уверены.

День 2: Прячемся от проблемы

На следующий день тех, кто заметил, что не все в порядке, уже больше, однако кто-то еще предполагает, что рыба могла просто устать. Еще не так сильно пахнет, чтобы люди зашевелились и начали действовать. Хотя покупателей поубавилось, но старший по должности пока ничего не сообщает хозяину, который, как всем известно, получив плохие новости, способен прийти в ярость.

День 3: Те же и «наладчик»

В какой-то момент на третий день, когда дохлая рыба действительно начинает вонять, управляющий вызывает хозяина, который в свою очередь вызывает независимого специалиста. Этот «наладчик» прибывает на место и громко объявляет, что рыба благополучно сдохла. Хозяин совершенно не согласен и ругает наладчика за то, что он не сделал рыбе искусственное дыхание. Когда наладчик указывает на бесполезность такого действия, ссылаясь на идущий от рыбы дух, хозяин требует от него «решить проблему рыбы».

Что он имеет в виду?

- Устранить запах?
- Предотвратить смерть тех рыб, которые еще не перемерли?
- Потребовать, чтобы поставщики привозили более здоровую рыбу?
- Найти покупателя на дохлую разлагающуюся рыбу?
- Все вышеперечисленное?

Понимая, что хозяин сам не знает, чего он хочет, и что его представление о том, что надо «исправить», изменится еще несколько раз, прежде чем его отпустят, «наладчик» закатывает рукава и приступает к работе. Тем временем бизнес идет все хуже.

Не замечать запах уже невозможно, и покупатели разбегаются. Распространяются слухи, что подохло очень много рыбы, которую тайком вы-

бросили. Кассандры жужжат: «Я же вам говорил!», а служащие стараются держаться отдохлой рыбы как можно дальше. Хозяин всех уверяет, что наладчик работает над проблемой и скоро найдет решение.

Тем временем наладчик по локоть залез в гниющие рыбы потроха, ему одиноко и тошно. Хозяин сообщает ему, какой он молодец. Поскольку это уже не первый раз, когда его призывают для решения «проблемы дохлой рыбы», наладчик слегка расстроен и недоволен тем, что им слишком «манипулируют». Но он знает, что добьется успеха, только если продолжит работу и найдет решение.

День 4: Поворотный момент

На четвертый день вонь уже так сильна, что работники требуют головы наладчика. Некоторые уже обвиняют его в том, что рыба сдохла. Кроме того, приезжает санитарная служба и грозит закрыть магазин, отчего хозяин впадает в панику.

Наладчик ко всему этому готов. Он знает, что день 4 – это момент истины: либо все благополучно решится, либо вонять будет вечно. Поэтому он не тратил время на искусственное дыхание для дохлой рыбы в день 3, зная по опыту, что ему понадобится время, чтобы исследовать внутренности и предложить решение. Сегодня перед ним несколько важных задач:

- Завершить фазу предварительного анализа
- Сформулировать план дальнейших действий
- Объявить о плане
- Начать его реализовывать
- Успокоить команду

Это трудные задачи, которых более чем достаточно для дня 4.

Он пока не пытается справиться с запахом (налаживать отношения с общественностью), понимая, что такие действия до принятия настоящего решения только приблизят катастрофу. Мысленно настройщик благодарит судьбу за то, что остальная рыба пока жива, потому что еще один труп опрокинул бы все его усилия.

День 5: Два критических маршruta

День 5 столь же важен, как день 4. С тех пор как сдохла первая рыбина, времени уже прошло много. Хотя труп убрали, запах остался. На-

стройщик знает, что если быстро не справиться с запахом, дело резко ухудшится. Пора доставать лизол. Он решает распределить свои силы между двумя задачами: продолжать поиски решения и избавиться от запаха. Ему приходится осторожно балансировать между обеими задачами.

Если он не станет изо всех сил искать решение, могут сдохнуть новые рыбы. Если он не уничтожит запах, хозяин может принять это за признак отсутствия прогресса и забраковать решение, которое предложит на-стройщик, так сказать, выплеснуть младенца вместе с водой.

Если настройщик сумеет пережить день 5, дальше будет легче. И работники, и покупатели скоро забудут про проблему, и он сможет завершить ее решение, не подвергаясь давлению с их стороны. Ему необходимо твердо и неуклонно двигаться вперед, следя, чтобы ни одна рыба не сдохла, пока он стоит на вахте. Важно не терять бдительности, пока кризис не преодолен полностью.

Мораль

Настройщики всего мира, помните, что в случае кризиса у вас есть три дня, чтобы показать себя:

1. Рыба начинает вонять на второй день, но вряд ли вас призовут так рано.
2. Не тратьте попусту времени в день 3. Вскрывайте рыбу и разрабатывайте план. Вонять будет сильно и долго.
3. Объявите свой план коллективу и начните его осуществлять в день 4.
4. Посвятите день 5 работе над проблемой и устранению запаха.

Удачи! Постарайтесь сохранить в живых всех рыбок, а если какие-то все же сдохнут, пусть они будут маленькими.

Резюме

Конечно, за три дня кризис не решить, но аллегория полезна для описания этапов, на которые нужно разбить работу. Помните, что когда вы прибудете на место, все – и ваш босс, возможно, тоже – будут в состоянии паники. Представьте себя техасским рейнджером, в одиночестве выходящим из вагона поезда, – к ужасу жителей, спасать которых он послан. «Мы дали телеграмму, что у нас бунт. Где же остальные?» – спрашивает старший из встречающих. «Один бунт – один рейнджер», – следует ответ. Вы должны сохранять спокойствие, когда все другие хватаются за оружие.

«Запах» у проекта может принимать разные формы. И распространители слухов о них позаботятся. В первое время одной из главных ваших задач будет определить главные причины запаха. Простая задержка выпуска программы? Не работают крупные блоки? Неудачный интерфейс пользователя? Нарушен процесс сборки? В достаточном ли объеме проводилось тестирование? В каком состоянии документация? Команда подняла открытое восстание? Маркетинг взбешен? Все эти и подобные запахи будут висеть в воздухе.

Избавиться от запаха – значит выполнить часть работы, относящуюся к «паблик рилэйшенз», чтобы остановить рост отрицательного отношения к проекту. Это означает необходимость спокойно объяснить, что некоторые из предполагаемых проблем просто не существуют или малозначительны. А что касается остальных действительных причин запаха, то вы должны очистить атмосферу, открыто рассказав о своем плане оздоровления.

Поэтому первая и главнейшая задача – *оценка*. Вы должны получить непосредственные данные о происходящем и произвести быструю и точную оценку людей, с которыми беседуете. Некоторые из них окажут вам помощь в выходе из кризиса, но не забывайте, что некоторые из них и были его причиной. Поэтому прикиньте, кто останется в стороне, а с кем вы пойдете дальше. В вашей команде должны быть такие игроки, которые согласятся с новым планом. Те, кто сильно сопротивляется переменам, окажутся слишком тяжелым грузом, чтобы быть полезными.

Оценивая обстановку, необходимо как можно быстрее разработать и начать реализовывать новый план. Это нужно сделать прежде всяких попыток ограничить ущерб. Вам просто нечего сказать, пока вы не можете объяснить, каким образом собираетесь изменить ход дел. Старательно избегайте всяких публичных оценок, пока не будете уверены в правильности сделанной вами оценки и не составите, хотя бы в общем виде, план, который можно обсуждать. Вы подвергнетесь сильному давлению, потому что люди хотят, чтобы их убедили в том, что «все будет хорошо». А сделать этого вы не сможете, пока ваш план оздоровления не будет готов.

Подготовив новый план, можно частично переключить внимание на уменьшение понесенного ущерба. Я не отрицаю, что это важно. Но заниматься этим эффективно можно только тогда, когда вы приложили все свои старания и выполнили две другие части своей работы.

ГЛАВА ДВАДЦАТЬ ВТОРАЯ

Расширение

Между расширением команды и ее продуктивностью существует важная связь. Есть очень простая модель, благодаря которой менеджеры могут лучше понять, что происходит, когда команда пополняется новыми членами. Модель проста, но позволяет делать некоторые любопытные предсказания. С помощью ряда простых диаграмм я покажу, как взаимодействуют различные факторы.

Невозможно четко ответить на все вопросы, связанные с расширением, но я уверен, что хороший менеджер может научиться управлять параметрами, определяющими модель. Выбор правильного соотношения между развитием организации (с фокусом на расширении) и управлением проектами (с фокусом на продуктивности) может принести большую выгоду. Для успеха и процветания организации необходимо эффективно решать проблемы ее роста и стремиться к росту продуктивности и снижению затрат в осуществляемых проектах. Эти две тенденции часто вступают в конфликт между собой и требуют тонкого соблюдения баланса. В этой главе я делаю попытку показать, что между ними общего, и способы, которыми могут быть достигнуты обе цели.

Проблемы роста

Успешные фирмы с течением времени расширяются. Попросту говоря, тем, кто успешно справляется с выполнением заказов, их дают все больше. Обычно расширение выражается в том, что в команду приходят новые люди.

Каждая успешная фирма вынуждена решать очень важную задачу, а именно оставаться успешной в условиях расширения. Задача сложнее, чем может показаться на первый взгляд. Необходимо успешно справляться с обычной работой. Кроме того, необходимо успешно справиться с новыми задачами. Наконец, что совсем немаловажно, и то и другое надо делать в условиях появления в команде новых членов, которые должны вписаться в организацию. А есть еще трудности *масштабирования*: то, что было относительно просто в условиях малой численности команды, становится более трудным при ее увеличении.

Даже в те периоды, когда фирма не расширяется, прием на работу новых сотрудников сопряжен с трудностями. Организации, лишенные периодического вливания «свежей крови», стагнируют, и в них возникает групповое мышление; а новые люди появляются всегда – просто потому, что необходимо компенсировать естественную убыль, о чем я вкратце скажу в конце главы.

В Кремниевой долине, месте, где сконцентрирована разработка ПО, встречается и другая проблема: у фирм есть разные причины, по которым они, я бы сказал, расширяются слишком сильно. Руководство видит, как возникают и быстро исчезают возможности, и дополнительные подрядчики и консультанты не всегда спасают. В попытке извлечь максимальную выгоду из своего лидерства или захвата плацдарма на новом рынке, фирмы сломя голову бросаются укреплять свои позиции за счет ускоренного роста: менеджерам среднего звена предлагается как можно больше расширять свои подразделения. Обычно результат оказывается далек от оптимального. Естественно, что, когда скорость расширения превышает 50% за год, должны возникать очень сложные проблемы.

Еще в 1972 г., когда политкорректность не была в такой моде, как сейчас, Фредерик Брукс, говоря в своей классической книге «Мифический человек-месяц» о добавлении новых людей в запаздывающие проекты, сформулировал Первый закон Брукса:¹

«Добавление людей в отстающий от графика проект приводит к еще большему отставанию».

Рассмотрим один из вариантов развития той же самой идеи. Новым будет только количественное обобщение для других мотивов к расширению

¹ Фредерик Брукс «Мифический человек-месяц, или как создаются программные системы». – Пер. с англ. – СПб.: Символ-Плюс, 2000.

организации – помимо безусловного рефлекса на отставание от графика. Иными словами, мы рассмотрим идею Брукса о том, что нельзя компенсировать нехватку «месяцев», увеличивая количество людей.

В примечаниях, сделанных Бруксом, есть два интересных числовых показателя. По оценке Высоцкого (Vyssotsky), крупные проекты выдерживают без ущерба прирост числа сотрудников, если он не превышает 30% в год. И в том же примечании цитируется Корбатто (Corbato), который указывает, что в длительных проектах следует ожидать ежегодной смены 20% сотрудников, что означает серьезную проблему интеграции новых людей (как и при расширении). Грань, отделяющая трудность завершения проекта продолжительными масштабными усилиями от неизбежности провала при слишком сильной миграции, узка (10%), и потому мы должны лучше – количественно – понять динамику появления новых людей в проектах.

Предыдущие работы в этой области, как правило, содержат качественные описания и относятся скорее к беллетристике, может быть, потому, что трудно получить надежные данные для построения соответствующих моделей. Редкое исключение составляет книга Шумана; у него более сложный подход, чем у меня, но это хорошее начало.¹

Наивная модель

Это простая модель «с одним периодом», в которой приняты следующие допущения:

- В начале периода команда хорошо интегрирована, хорошо обучена и работает в полную силу. Продуктивность ее членов мы возьмем за основу для сравнения.
- По ходу дела мы добавляем новых участников примерно с такой же квалификацией, как и у тех, кто занят в проекте с самого начала.
- В конце периода мы имеем «новую» команду большей численности и с большими возможностями для выполнения работы, потому что людей с той же средней продуктивностью стало больше, чем было вначале.

Нас здесь интересует, что происходит с продуктивностью в *переходный период*.

¹ Shooman, Martin L. Software Engineering: Design, Reliability, and Management (New York: McGraw-Hill, 1983), p. 469–479.

Обратите внимание на важное допущение: мы предполагаем, что общая продуктивность *на одного работника* не уменьшается из-за роста команды. Поскольку часто бывает *не так*, мы наивно предполагаем, что, как только закончится переходный период, общая продуктивность команды увеличится пропорционально росту числа сотрудников.

Эта модель корректна и для команд, численность которых постоянно увеличивается. Мы просто увяжем несколько однопериодных моделей в единую последовательность. Путем суперпозиции можно делать предсказания и для этого случая. Математически это немного сложнее, но принципиально не добавляет ничего нового.

Влияние новых работников в переходный период: полезный вклад

Даже в переходный период необходимо увеличить общую продолжительность эффективно расходуемого рабочего времени. Даже если эффективность работника в период его становления составляет всего 10%, его вклад за неделю составит четыре часа «полезной» работы. Оставшиеся 36 часов следует рассматривать как «обучение», «подготовку», «инвестиции в будущее». Они оплачиваются, но в конечном продукте не отражаются.¹

Итак, в этой наивнейшей модели мы в переходный период выигрываем время (приобретаем часы), но проигрываем в производительности. Это разумный компромисс, и мы должны оценить его количественно, чтобы знать, что на что мы меняем. Однако эту модель необходимо уточнить, приняв в рассмотрение вторичный эффект, имеющий здесь место, а именно торможение.

Влияние новых работников на действующую команду: торможение

Оказывается, что наши новые работники хотя и добавляют рабочие часы, поскольку на 10% продуктивны, их участие все же снижает общий результат. Причина в том, что остальные члены команды теряют время, зани-

¹ В центре внимания у меня находится разработка ПО. Однако этот анализ применим и к другим типам команд. Например, если команда занимается продажами, то слова «часы разработки ПО» надо заменить на «часы непосредственных продаж». Между всем остальным также могут быть установлены аналогии. При добавлении новых торговых агентов поступления растут, но вместе с ними растут и издержки торговли на одного работника.

маясь с новичками. Последние нуждаются в контроле, им надо многое объяснить, «показывать хитрости», *исправлять их ошибки* т. д., пока они не достигнут пика производительности. Это время стоит очень дорого, поскольку производительность опытных работников равна 100% по определению. Иными словами, вновь принятые сотрудники тормозят команду, потому что часть сил опытных работников, обладающих 100-процентной производительностью, расходуется на взаимодействие с новыми.

Должен отметить ряд очевидных исключений из этого правила. Если мы нанимаем нового работника, обладающего нужными нам умениями, которые отсутствовали до его появления, общая продуктивность может самопроизвольно вырасти. Аналогично при найме старшего менеджера может заработать команда, до этого не показывавшая особых успехов. Оба эти случая служат примером почти немедленного улучшения и представляют собой, так сказать, «отрицательное торможение». Но в целом они составляют исключение.

Модель и действующие в ней допущения

Как и в любой модели, мы сделаем некоторые допущения, которые для удобства я соберу в один список. Обратите внимание, что для повышения точности можно строить все более сложные модели, но в какой-то момент мы столкнемся с законом убывающего плодородия. Самые лучшие модели самые простые.

Мы сделаем такие допущения:

- **Уже работающие участники команды «продуктивны на 100%».** Какое-либо снижение их продуктивности, связанное с обучением или накладными расходами, мы игнорируем. На них мы будем основывать исходную оценку продуктивности и все расчеты. Мы предполагаем, что суммарная продуктивность команды равна произведению средней продуктивности работников на их численность в стабильном состоянии. Мы не предполагаем, что продуктивность всех работников одинакова, но считаем, что она как-то распределена и может быть охарактеризована средним значением.
- **В период своего становления (один год) новые работники обладают некоторой средней продуктивностью P , лежащей в диапазоне $0 \leq P \leq 1$.** Для простоты мы полагаем, что их набирают всех сразу в начале года. Например, если $P = 0,6$, то в течение первого года средняя продуктивность новых работников составит 60% от

продуктивности уже имеющихся членов команды: 60% их рабочего времени уходит на «полезную» работу над продуктом, а 40% – на «набор скорости». Можно, конечно, без особого ущерба для целостности картины, выбрать любую продолжительность периода становления; по моим наблюдениям он обычно оказывается дольше, чем предполагают менеджеры.

Несложное развитие модели могло бы включить наличие «кривой обучения». Иными словами, можно, поместив данные в электронную таблицу, смоделировать обычную S-кривую, которая охарактеризует успехи обучения новобранцев. Но ради простоты я сделал допущение о том, что средняя продуктивность на протяжении всего периода постоянна.

- **Мы измеряем ежегодное увеличение численности команды как часть G , которую составляют новобранцы (обычно между 0 и 1).** Например, $G = 0,1$ обозначает прирост в 10%, т. е. появление одного нового члена команды на каждые 10 уже работающих. $G = 1,0$ соответствует приросту в 100% или попытке добавить одного нового участника на каждого имеющегося. Как мы увидим далее, такие скорости прироста связаны с высоким риском. Еще раз повторю, что мы делаем упрощающее предположение, что новые члены команды принимаются на работу все сразу в начале года. В более сложных моделях поступление новых работников происходит поэтапно. Это приводит к более сложным вычислениям, но не представляет ничего принципиально нового. Такая модель может быть построена с помощью электронной таблицы.
- **Эффект от принятия на работу новых людей может быть охарактеризован коэффициентом торможения D , где $0 \leq D \leq 1$.** Например, $D = 1,0$ означает, что на каждый час непродуктивной работы нового члена команды приходится потеря одного часа работы старого члена команды. Чем меньше D , тем лучше для организации; например, $D = 0,2$ означает, что мы теряем всего два часа работы старых членов команды на каждые 10 часов потеряннного труда новичков.
- **У новых членов команды такой же тариф почасовой оплаты труда, как у старых.** Это разумно, поскольку мы считаем, что, как только закончится период вставания, новые члены будут работать так же продуктивно, как старые. То есть мы предполагаем, что они достигнут «стопроцентной» продуктивности. Иными словами, мы

предполагаем, что вновь принятые работники в среднем обладают такой же квалификацией, как уже имеющиеся, поэтому им платят по тому же среднему тарифу.

Следствия из этой модели

Сначала я хочу выяснить, удастся ли в результате получить больше итоговых часов работы над продуктом. Здесь конкурируют два фактора:

- С одной стороны, мы получаем новые часы благодаря продуктивности новых членов, какой бы низкой она ни была.
- С другой стороны, их обучение оборачивается потерей продуктивности части уже имеющихся членов команды.

Уравнение, связывающее H , количество полезных часов, с G , P и D , имеет следующий вид:

$$H = 1 + G [P - (1 - P) D].$$

Сделаем несколько быстрых проверок:

- При $G = 0$ получаем $H = 1$. Разницы в рабочих часах нет.¹
- При $P = 0$ получаем $H = 1 - DG$. Полезные часы сокращаются на величину торможения для всей организации.
- При $P = 1$ получаем $H = 1 + G$. Полезные часы растут со скоростью расширения.
- При $D = 0$ получаем $H = 1 + PG$. Прирост продуктивных часов равен скорости расширения, помноженной на продуктивность новых членов команды.
- Наконец, при $D = 1$ мы видим, что $H = 1 + G (2P - 1)$. Каждому часу, потраченному новыми участниками команды на освоение работы, должен соответствовать один час времени старого участника команды. Если значение P не достигнет хотя бы 0,5, мы не окупим времени, потраченного старым работником на обучение нового.

На самом деле количество полезных часов, потраченных на работу над продуктом, увеличится тогда и только тогда, когда

$$P > (1 - P) D.$$

Обратите внимание, что это условие не зависит от скорости расширения G . Иными словами, всегда можно получить дополнительные часы пу-

¹ Как нет и новых членов команды. – *Примеч. науч. ред.*

тем более быстрого расширения, если выполнено это соотношение между P и D .

Ищем коэффициент

Уравнение

$$H = 1 + G [P - (1 - P) D]$$

можно переписать так:

$$H = 1 + GM,$$

где

$$M = P - (1 - P) D.$$

На рис. 22.1 отображена величина M , которая при умножении на коэффициент роста G дает процентное увеличение полезных часов работы над продуктом. (Поскольку в качестве исходного значения выбрана 1, величина « GM » фактически представляет увеличение «сверх единицы», т. е. процентный рост.)

- Пять белых полосок слева соответствуют *отрицательным* коэффициентам. В этой области мы видим *сокращение* количества полезных часов на единицу роста; более быстрый рост в данном случае приводит к еще большему сокращению времени, остающегося для работы над продуктом.
- Остальные окрашенные полосы представляют собой последовательно улучшающиеся сценарии; например, первая полоска рядом с белой областью соответствует значениям M от 0 до 0,2. В этой области мы получаем для прироста полезных часов среднее значение «0,1 умножить на коэффициент роста»; например, при росте в 30% мы получим прирост полезных часов $0,1 \times 30\%$, или 3%.
- Другая крайность – темная полоса в правой части диаграммы, соответствующая значению M от 0,8 до 1,0, поэтому в данной области можно принять среднее значение равным 0,9. Для тех же 30% роста численности мы теперь получаем $0,9 \times 30\%$, или на 27% больше полезных часов для работы над продуктом.

Подводя итоги, мы видим, что этот коэффициент нелинейным образом зависит от двух факторов:

- P служит характеристикой поступившего пополнения работников; это оценка их относительной продуктивности в период становления.

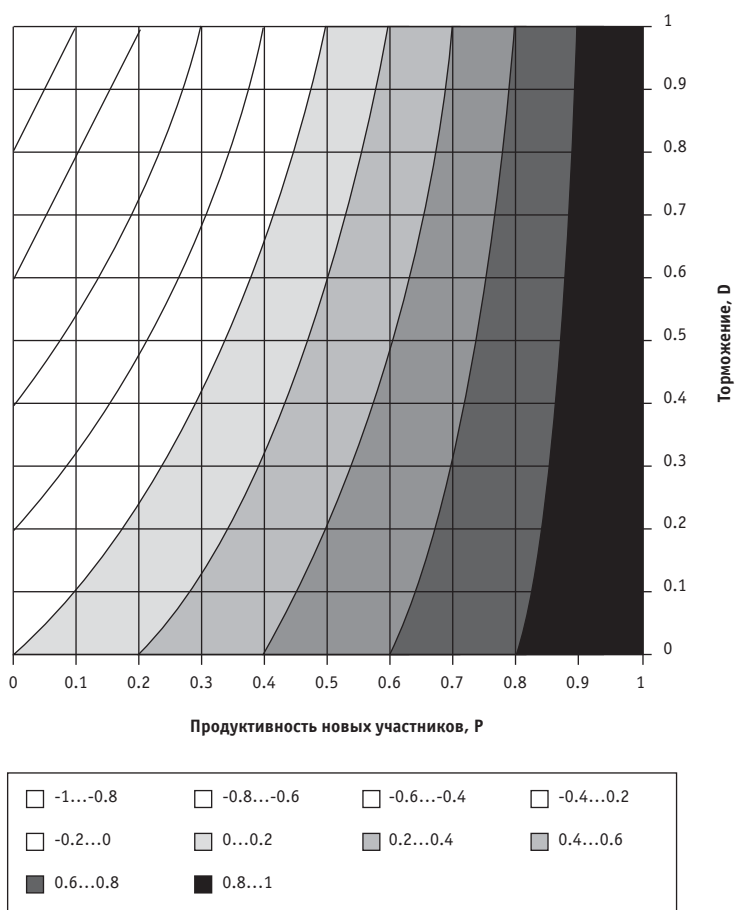


Рис. 22.1. Семейство кривых коэффициента M как функции торможения D и продуктивности P

- D характеризует нашу нынешнюю организацию; это мера нашей способности эффективно включить в работу новых людей.

Из этой диаграммы видно, что если удастся удержать D на небольшом уровне, то приемлемыми окажутся даже довольно низкие значения P . С другой стороны, чем выше D , тем более чувствительны мы оказываемся к P . И это не противоречит здравому смыслу.

Большие времени для работы над продуктом

Разобравшись с действием множителя, мы можем построить простой график (рис. 22.2), иллюстрирующий процентный рост полезных часов в зависимости от скорости расширения. Он оказывается просто линейным. Зная значение M , можно узнать процент прироста полезных часов, найдя значение G на горизонтальной оси и выбрав наклонную линию для соответствующего M . Результат находится по вертикальной оси. Можно поступить наоборот: для нужного процента увеличения полезных часов найти скорость расширения, которая необходима при заданном значении M .

А во что это обойдется?

Не забывайте, что вам придется оплатить все рабочее время, продуктивно оно или нет. Как только появляются непродуктивные часы (из-за новых членов или вызванного ими торможения), общая продуктивность организации падает.

Руководству следует определить разумный компромисс. Вы создадите продукт *быстрее*, потому что выделите на него больше рабочего времени, но при этом создание продукта обойдется *дороже*, потому что общая продуктивность организации уменьшится из-за необходимости обучения новых работников. Руководство должно решить, на какие издержки можно

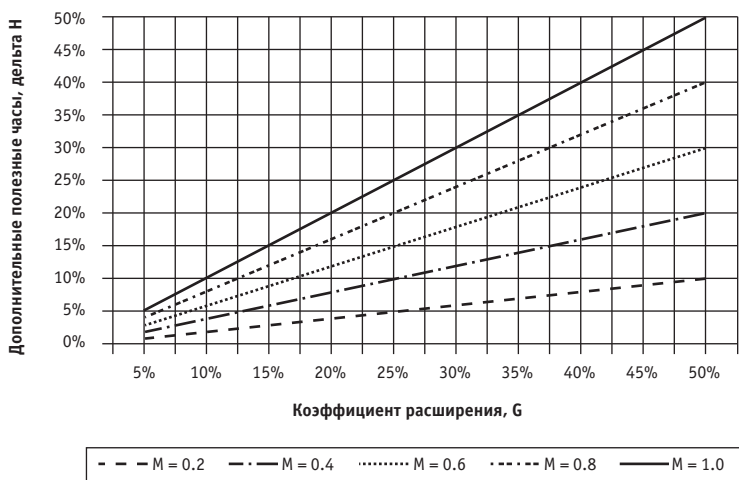


Рис. 22.2. Добавившиеся к проекту полезные часы ΔH как функция коэффициента расширения G при разных значениях M

пойти ради ускорения работы. Во многих проектах, особенно программных, *оплата труда составляет большую часть стоимости*, поэтому ее нельзя не учитывать.¹

Суммарное количество полезных часов, выделенных на разработку продукта, составит

$$H = 1 + GM.$$

С другой стороны, суммарное количество оплачиваемых часов составит

$$T = 1 + G.$$

Таким образом, новая величина продуктивности организации будет равна

$$N = H / T,$$

или

$$N = (1 + GM) / (1 + G).$$

Можно построить график этой зависимости и посмотреть на изменение общей продуктивности как функции M и G . Он приведен на рис. 22.3.

Справа находится «хорошая» зона. Это значит, что продуктивность там составляет 95–100% от исходной. На другом конце наблюдается падение общей продуктивности до 70–75% от исходной. Обратите внимание, что лучшая защита от стремительного падения продуктивности – это постепенный рост, скажем на уровне 20%. Сохраняя рост не выше 20%, мы гарантируем, что общая продуктивность не упадет ниже 85% независимо от значения множителя; она также не упадет ниже 90%, пока M имеет значение хотя бы 0,4. С другой стороны, при росте 20% и выше мы попадаем в большую зависимость от M .

Попутно отметим, что если удастся удержать M на уровне 0,85 или выше, то можно остаться в зоне 95-процентной продуктивности независимо от скорости расширения. Достичь для M значения 0,85 – это не простая задача, как видно из рис. 22.1.

А что же стоимость труда, необходимого для выпуска продукта? Если в единицу времени вкладывать в производство продукта H часов полезно-

¹ Конечно, при этом надо еще гарантировать, что все происходит в условиях «равного или лучшего качества». Ускорять выпуск на рынок продукта за счет увеличения его стоимости может оказаться неоправданным, если при этом пострадает его качество.

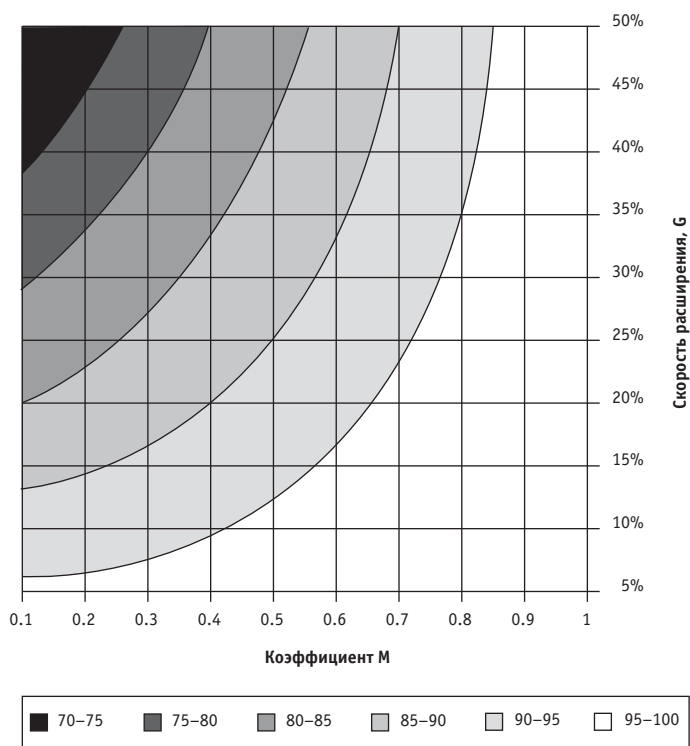


Рис. 22.3. Семейство кривых для новой суммарной продуктивности (в процентах от исходной) как функции скорости роста G и коэффициента M

го времени, то продукт будет завершен за время $1/N$. Но каждая единица времени теперь обходится нам в $(1 + G)$ долларов, поэтому новая стоимость составит $(1 + G) / N$. Это просто величина, обратная общей продуктивности. Представление о том, что стоимость и продуктивность находятся в обратной зависимости друг от друга, согласуется с нашими представлениями о здравом смысле.

На рис. 22.4 показано, как растет стоимость продукта в зависимости от M и G .

Наглядный пример

Предположим, что мы тратим на оплату труда 1 миллион долларов в год и рассчитываем, что сможем выпустить новый продукт к концу

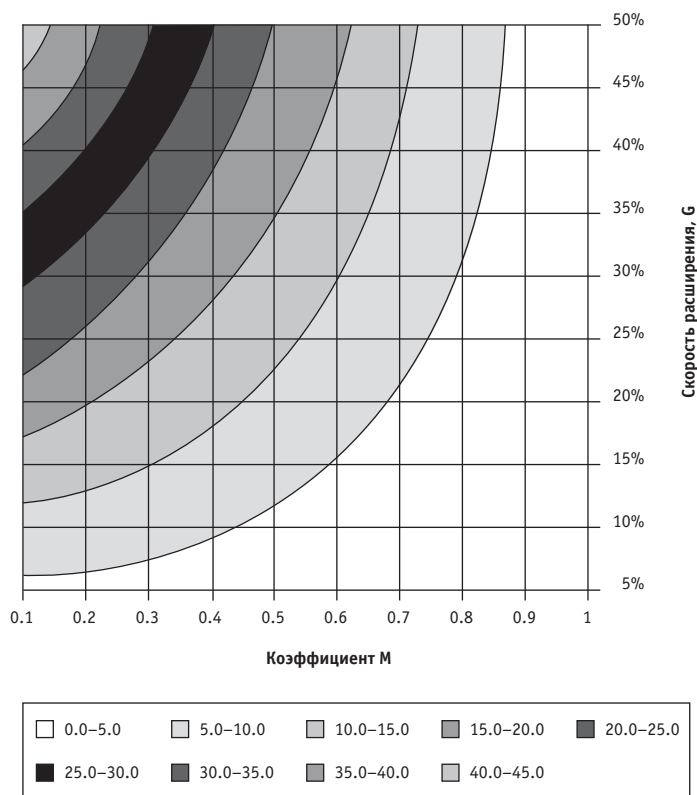


Рис. 22.4. Семейство кривых процентного увеличения стоимости продукта как функции скорости роста G и коэффициента M

года с той командой, которая у нас есть. Руководство желает получить готовый продукт не через 12 месяцев, а через 10. Мы предполагаем, что все вновь принятые на работу обеспечат в течение первого года лишь 60% продуктивности и что средний коэффициент торможения для нашей организации D составит 0,5. Нам надо определить:

- Сколько новых сотрудников следует принять на работу?
- Как это отразится на стоимости разработки продукта?

Чтобы создать за 10 месяцев то, на что отведено 12, надо выделить на продукт на 20% больше времени (полезных часов). (Проверим: 10 месяцев умножить на 1,2 часа/месяц дает эквивалент того, что прежде должно было быть произведено за 12 месяцев.) Поэтому $H = 1,2$, а это значит, что $GM = 0,2$.

Но $M = P - (1 - P)D = 0,6 - (0,4)(0,5) = 0,6 - 0,2 = 0,4$. Таким образом, G равно $0,2/0,4$, или $0,5$, что означает рост численности на 50%.

Стоимость труда, вложенного в продукт за единицу времени, вырастает на 50%, потому что $G = 0,5$. Мы изготовим его всего за $(^{10}/_{12})$ предполагавшегося времени, поэтому общая стоимость возрастет на 25%: поскольку мы теперь тратим на оплату труда \$1,5 миллиона в год, но выпустим продукт за $(^{10}/_{12})$ г., его суммарная стоимость составит \$1,25 миллиона. Обратите внимание, что увеличение на 25% точно совпадает с тем, что предсказывает график на рис. 22.4.

Кроме того, по графику суммарной продуктивности на рис. 22.3 при M , равном $0,4$, и G , равном $0,5$, мы получаем суммарную продуктивность 80%. Это согласуется с выводом о том, что, тратя на продукт 1,2 полезных часа, мы платим за 1,5 полных часа; $1,2$ делить на $1,5$ дает 80%. Величина, обратная к $0,8$, составляет $1,25$, что согласуется с результатом, согласно которому стоимость продукта возросла на 25%.

Подведем итоги. Чтобы сократить 12-месячный график на два месяца, мы должны поднять численность занятых на 50% и увеличить стоимость продукта на 25%. Дело в том, что продуктивность вновь набранных людей составляет лишь 60% от текущей, а на каждые два часа их обучения опытный участник команды тратит один час своего времени. В дополнение к этому, 50-процентная скорость роста подвергает нас высокому риску.

Расчет последствий намерения руководства выпустить продукт на рынок через 9 месяцев вместо 12 оставляем читателям в качестве упражнения. Интересно посмотреть, насколько увеличатся стоимость проекта и риск в результате сжатия графика еще на один месяц.

Нелинейность

Обычно интуиция не подводит нас, когда зависимость от параметров носит линейный характер. Например, при прочих равных условиях мы рассчитываем, что увеличение объема работы втрое увеличит время ее выполнения. Однако интуиция не так хороша, когда переменные связаны между собой нелинейным образом.

В данной конкретной задаче мы замечаем, что после вычисления множителя M связь между количеством дополнительных полезных часов работы над продуктом и скоростью роста G становится линейной. То есть прирост полезных часов находится в прямо пропорциональной зависимости

от скорости роста с коэффициентом пропорциональности, равным M . К сожалению, во всей модели это единственная линейная зависимость.

Для вычисления главной составляющей, множителя M , мы воспользовались нелинейной зависимостью от переменных P и D :

$$M = P - (1 - P) D$$

Это приводит нас к рис. 22.1, на котором разные полосы соответствуют разным диапазонам M . Обратите внимание, что интуиция дает весьма слабое представление о том, какой функциональной зависимостью связаны эти полосы с P и D : мы знаем, что малое P и большое D оказываются плохой комбинацией, но определить, насколько плохой, трудно. Вот почему график оказывается таким полезным.

Аналогично общая продуктивность и возросшая стоимость создания продукта нелинейно зависят от M и G . Вспомним, как выглядит соотношение для новой общей продуктивности:

$$N = (1 + GM) / (1 + G).$$

Очевидно, мы имеем еще одну нелинейную зависимость. Мы не очень хорошо представляем себе, как ведет себя общая продуктивность, если не считать замечания, что плохо, когда G велико, а M мало. Увеличение стоимости является величиной, обратной к N , следовательно, эта зависимость тоже нелинейная. Графики этих величин на рис. 22.3 и 22.4 имеют такой же полосчатый вид, как при вычислении M . А поскольку M нелинейно зависит от P и D , мы получаем одну нелинейность, наложенную на другую! Поэтому попытка сделать вывод о поведении N как функции P , D и G представляет собой трудное упражнение: зависимость, которую мы хотим интуитивно оценить, выглядит так:

$$N = (1 + G [P - (1 - P) D]) / (1 + G)$$

Большинство из нас вынуждено будет согласиться, что интуиция едва ли что-нибудь подскажет нам о поведении N как функции этих трех переменных. Тем не менее, это именно та задача, с которой мы и начали.

Мораль этой истории такова:

Даже простые природные явления иногда оказываются *нелинейными*.

Когда это случается, интуиция мало помогает нам. В понимании происходящего и способности делать разумные предсказания неоценимую помощь может оказать простая математическая модель и некоторые средства графического отображения.

Призыв к действию

Мы разобрались, как различные взаимодействующие факторы влияют на продуктивность в период расширения организации, и должны рассмотреть вопрос о принятии каких-то мер. У меня есть три предложения:

- **Если есть возможность, следует воспользоваться реальными данными для определения параметров Р и D.** Это необходимые составляющие для вычисления множителя М, от которого зависит все остальное. Если брать их значения с потолка, можно сильно просчитаться. Не во всех организациях ведется достаточно подробный учет, который мог бы вам помочь. С другой стороны, можно посмотреть, например, сколько часов явно отводится на обучение. Эти цифры учитываются и в Р, и в D, поскольку касаются и новых, и старых работников. Надо просмотреть протоколы всех совещаний и выяснить, чему они были посвящены в большей мере – разработке продукта или подготовке вновь принятых на работу. Чем точнее вы оцените Р и D, тем более надежные цифры получатся в итоге.
- **Не забывайте про Р, оценивая кандидатов для приема на работу.** Модель очень чувствительна к Р, и его низкое значение нельзя компенсировать ничем.
 - Следует брать тех, кто способен быстро учиться. У тех, кто медленно учится, низкий Р, и их следует избегать.
 - Остерегайтесь тех, у кого отсутствует какой-то навык, например знание конкретного языка программирования или технологии, которые якобы легко освоить. Даже самым умным требуется какое-то время, чтобы изучить новый материал; если человеку придется учиться в рабочее время, значит, вы согласны, что его Р будет ниже.
 - Избегайте принимать людей из компаний с «культурной средой», сильно отличающейся от вашей. У тех, кому приходится дополнительно адаптироваться к тому, «как это делают здесь», неявно, но иногда существенно снижается Р (и соответственно возрастает D).
 - Если у кандидата отсутствует гибкость, то все эти факторы усиливаются еще больше.
- **Можно считать, что D усиливает эффект малого Р.** Любопытно, что в маленьких организациях D бывает выше, чем в более солидных. Дело в том, что большая часть всей организационной премудрости хранится в головах лишь нескольких людей. Когда приходят новые

люди, нет иного способа ввести их в курс дела, кроме общения один на один с этими ключевыми фигурами. Очевиден ряд вещей:

- Набирайте и обучайте новых людей партиями, что даст некоторую экономию масштабирования в процессе адаптации.
- Старайтесь как можно больше документировать свои производственные технологии, чтобы новички могли прочесть материал, а не отрывать людей от работы по каждому вопросу.
- В организациях, занимающихся разработкой ПО, наличие хорошо написанного и хорошо документированного кода может резко уменьшить D , а также способствовать увеличению R .

«Серебряной пули» здесь не существует, но понимание важности как R , так и D , может помочь менеджерам свести к минимуму возможные потери. Оно позволяет также оценить окупаемость таких вещей, как составление документации. Уменьшив D всего на несколько десятых, мы можем увидеть, насколько улучшится общая продуктивность и сократятся издержки роста при очередном расширении организации. А потом сравнить расходы на такое сокращение D с приобретаемыми взамен выгодами.

Выводы

Я исследовал очень простую модель для расширения и продуктивности организации во время переходного периода при приеме новых работников с тремя параметрами. Я моделировал «полезные часы» и «суммарную продуктивность организации» как функцию скорости роста, продуктивности новых работников и эффективности ассимиляции их организацией. Несмотря на свою простоту, эта модель показывает, что большие темпы роста связаны с риском, а сочетание быстрого увеличения численности с низкой продуктивностью новых работников может оказаться губительным. Это происходит даже тогда, когда установившаяся продуктивность новых работников столь же высока, как и у прежних. Если в придачу организация окажется неэффективной в ассимилировании новых членов, то катастрофа почти неизбежна.

Даже если вновь принятые на работу так же хороши, как те, кто уже работает, есть и другая опасность: результирующая более крупная организация может оказаться менее продуктивной в пересчете *на одного работника*, чем прежняя. Продуктивность может пострадать из-за эффектов размера и масштабирования, например роста накладных расходов или роста числа каналов связи, и т. д. В нашей модели эти эффекты игнорировались.

В реальности могут происходить еще более коварные вещи: стремясь к быстрому расширению, мы часто снижаем требования к принимаемым на работу. Когда это происходит, не только тяжелее оказываются объединенные потери «расширение + обучение», но и разбавляется капитал талантов, что ведет к устойчивому снижению продуктивности даже после завершения процесса адаптации новых работников. Перед организацией встает сложная проблема: во время периода адаптации мы приписываем падение продуктивности действиям, связанным с адаптацией, и только когда этот период завершен, мы замечаем, что пострадала наша общая продуктивность. Замечаем, конечно, слишком поздно. Долгосрочная проблема ускользнула от нас, скрывшись под маской временной или краткосрочной. Вот почему так важно при расширении организации набирать таких людей, которые в установившемся режиме могут поднять среднюю продуктивность группы.

Попутно мы сделали интересное наблюдение: несмотря на свою простоту, наша модель привела к нескольким нелинейным зависимостям. Наши инстинкты плохо приспособлены к нелинейным явлениям. Графики, с помощью которых мы проиллюстрировали взаимозависимости переменных, вскрыли неожиданные зависимости. Вот такие «скрытые» зависимости могут оказаться сюрпризом для неплохих в общем-то менеджеров: из-за нелинейности расхождение с ожиданиями оказывается очень резким. Попытка сократить сроки проекта всего на один месяц путем привлечения к нему новых людей может привести к катастрофическим последствиям, если вы уже находитесь в «желтой зоне», не зная этого. Сделав этот последний маленький на первый взгляд шаг, вы срываетесь в пропасть. Лучше нарисовать несколько графиков и попытаться выяснить, где вы находитесь в действительности, чем с криком лететь вниз.

Следует обратить внимание на сделанные мной предложения для улучшения параметров P и D , управляющих всем остальным поведением. Это «призыв к действию». Модели нужны только для того, чтобы с их помощью понять природу явления, а потом постараться изменить значения параметров, отрицательно влияющих на нашу деятельность.

В модели сделан ряд упрощающих предположений, о которых я благо разумно постарался вас предупредить. Если, исходя из этих предположений, модель предсказывает неблагоприятные тенденции продуктивности и затрат, менеджеру надлежит действовать с осторожностью. Причина проста: природа редко бывает столь же добра к нам, как математическая модель. Когда свою руку к ситуации прикладывает еще мистер Мерфи (из-

вестный «законом Мерфи»), можно не сомневаться, что результаты в целом окажутся *хуже*, чем предсказывает наша модель. Необходимо оставлять люфт для возможной ошибки. Поэтому совет «относиться к модели с известным скептицизмом» означает следующее: если модель говорит, что все в порядке, двигайтесь с осторожностью; если модель предсказывает проблемы или высокий риск, вернитесь и просчитайте все заново. Как правило, менеджеры проявляют чрезмерный оптимизм в оценке параметров, входящих в модель, что в сочетании с некоторыми «идеализирующими» допущениями может привести к тяжелым последствиям.

Номограмма

Было бы прекрасно, если бы вычисления, как в рассмотренном выше наглядном примере, можно было проводить быстро, а еще лучше – не подставляя числа в уравнения. Соотношения просты, но даже я должен признать, что работа с уравнениями не самое любимое занятие большинства современных менеджеров. Однако не все потеряно.

В те времена, когда корабли были из дерева, а люди из железа, имелось простое решение для отражения связей между переменными: графические методы и, в частности, такой инструмент, как *номограмма*.¹ Пользователь просто накладывал поверочную линейку на некоторые особым образом расположенные и калиброванные шкалы. Если требовалась серия вычислений, несколько таких чертежей соединялись в цепочку на одном листе бумаги. Нетривиальные математические расчеты, необходимые для номограммы, выполнял инженер, а мастер-специалист по номограммам вычерчивал диаграмму, пользоваться которой было на удивление просто. Эта технология, описанная д’Оканем около 1900 г., активно разрабатывалась мировым сообществом инженеров в 1925–1975 гг., пока появление карманных калькуляторов и компьютеров не сделало ее немодной. Четверть века спустя, как часто бывает с полезными вещами, попадающими на обочину истории, мы ждем для нее достойной замены. Боюсь, что напрасно.

Я создал такую номограмму для уравнений этой главы. Она приведена на рис. 22.5. Как во всех хороших номограммах, прямо на графике приведен пример ее применения. Обратите внимание, что это уже знакомый нам наглядный пример.

У номограмм есть интересное свойство – они решают уравнения явно и неявно. Любую отсутствующую переменную или их комбинацию можно

¹ От греческого *nomos* (закон) и *graphein* (писать).

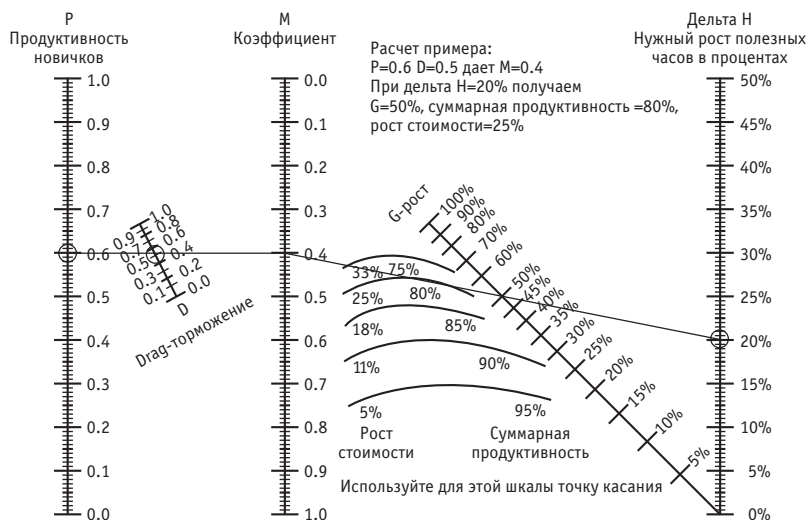


Рис. 22.5. Номограмма роста и продуктивности

найти «обратным движением». Поэкспериментируйте немного. Например, начните с «допустимого» прироста стоимости и желательного процента прироста полезных часов, чтобы найти необходимую скорость расширения и соответствующий множитель. Потом посмотрите, какие комбинации R и D дают вам это значение M . Часто подобные исследования позволяют выяснить, можно ли вписаться в заданные ограничения.

Я расскажу, как построил эту номограмму. Первая часть (если слева направо) основана на том наблюдении, что связь между R , D и M имеет стандартный вид, поддающийся представлению номограммами «Z-типа»; нелинейная шкала для D строится вручную по соответствующим общим точкам пересечения. Затем добавляются номограммы M , дельта N и G (полезные часы получаются умножением прироста G на коэффициент M). Здесь снова применима немного иная номограмма «Z-типа», а нелинейная шкала G определяется методом пересечений. Наконец, когда мы знаем, какая дельта N нам нужна, и коэффициент M , который уже определили, можно узнать общую продуктивность и обратную к ней величину – увеличение стоимости. Они, однако, оказываются огибающими, а не просто пересечениями. При графическом построении их были использованы пары значений дельта N и M , соответствующие каждой величине общей продуктивности; добавление прироста стоимости состоит в том, что в нужном месте ставится метка.

Электронная таблица

Более современный подход, имеющий целью упростить вычисления, действует электронную таблицу. На рис. 22.6 приведен скриншот такой таблицы (ее я построил намного быстрее, чем построил бы номограмму).

Начав с этой очень простой таблицы, вы можете ввести более сложные усовершенствования, о которых говорилось выше в этой главе. Например, я дал возможность задавать разный коэффициент оплаты для вновь принятых на работу, хотя в примере зарплаты были равны, поэтому можете

В такие ячейки введите данные:	<input type="text"/>	Пример
В таких ячейках появятся результаты	<input type="text"/>	
Введите количество человеко-часов для проекта:	<input type="text"/>	7000
<u>Исходные данные</u>		
Введите текущую численность команды	<input type="text"/>	5
Введите средний почасовой тариф \$/час	<input type="text"/>	50
При <input type="text"/> часах в рабочей неделе		40
Проект потребует <input type="text"/> 35 недель		35
Общая стоимость проекта составит <input type="text"/> \$ 350,000		\$ 350,000
<u>Вновь принятые на работу</u>		
Введите количество принятых:	<input type="text"/>	1
Введите средний почасовой тариф \$/час	<input type="text"/>	50
Введите относительную продуктивность P:	<input type="text"/>	0,6
Введите торможение D:	<input type="text"/>	0,1
<u>Результаты</u>		
Коэффициент роста:	<input type="text"/> 20%	20%
Полезных часов в неделю благодаря принятым на работу:	<input type="text"/> 24	24
Часов в неделю, потерянных из-за торможения:	<input type="text"/> 1,6	1,6
Чистых часов в неделю благодаря принятым на работу:	<input type="text"/> 22,4	22,4
Суммарно часов в неделю для работы над проектом:	<input type="text"/> 222,4	222,4
Суммарная продуктивность:	<input type="text"/> 92,7%	92,7%
Теперь проект займет <input type="text"/> 31,5 недель		31,5
Что составит экономию <input type="text"/> 3,5 недель		3,5
График сокращен на <input type="text"/> 10%		10%
Новая стоимость проекта <input type="text"/> \$ 377,698		\$ 377,698
Увеличение стоимости проекта на <input type="text"/> \$ 27,698		\$ 27,698
Процент роста стоимости проекта <input type="text"/> 7,9%		7,9%
Коэффициент M <input type="text"/> 0,56		0,56

Рис. 22.6. Электронная таблица роста и продуктивности

сравнить результат таблицы со своими расчетами с помощью уравнений или номограммы. Кроме того, когда модель преобразована в форму электронной таблицы, можно воспользоваться специальными инструментами типа Goal Seeking для неявного решения или движения в обратном направлении, если это вам потребуется.

Резюме

Материал этой главы первоначально появился в «The UMAP Journal», 25 (4), (2004) 357–374.

Вот еще одно маленькое замечание по поводу естественной убыли (ее эффект может быть коварным). Предположим, например, что в вашей группе 20 человек и четверо из них ушли. Естественная убыль составила 20%. Чтобы вернуть себе прежние позиции, надо нанять четырех человек. Однако это соответствует росту численности в 25%, потому что надо добавить четырех человек к имеющимся 16. При учете естественной убыли следует помнить, что коэффициент роста G , который понадобится для вычисления новой суммарной продуктивности, всегда будет выше коэффициента естественной убыли, что и демонстрирует данный пример.

ГЛАВА ДВАДЦАТЬ ТРЕТЬЯ

К

Культура

В настоящее время наблюдается возрождение интереса к культуре и ценностям в бизнесе,¹ что я рассматриваю как еще один пример нетерпимости природы к пустоте: культура и ценности возвращаются в центр внимания потому, что мы недостаточно интересовались ими в последнее время. В то время как продолжают множиться известия о заблудших компаниях и их сотрудниках, те, кто выбрал праведный путь, обычно оказываются в забвении. Так всегда бывает: слабостями и причудами мошенников заполнены страницы газет и эфирное время, а честным людям достается мало славы, даже если они в большинстве.

В этой главе мы изучим связь между «культурой» и «ценностями». Я горжусь тем, что работал в компании с крепкой культурой, которая соединялась с правильными ценностями. Но это не случайность. Для меня несомненно, что мой собственный выбор ценностей довольно близко соответствует тому, который существует в Rational Software, иначе я не провел бы там более 16 лет.

Начнем с изучения вопроса о том, что же такое культура, а потом посмотрим, какую роль играют ценности в определении и поддержке культуры.

¹ См., например, Amy C. Edmondson, Sandra E. Cha, «When Company Values Backfire». *Harvard Business Review*, November 2002, или Patrick Lencione, «Make Your Values Mean Something». *Harvard Business Review*, July 2002.

Что такое культура?

Я определяю *культуру* как набор характеристик, по которым одна группа людей отличается от другой. Например, иногда мы рассматриваем несходство в поведении англичан и французов как проявление их культурных различий. При этом различий много, но все они объединяются под одним названием.

Некоторые характеристики не связаны с отдельными культурами. Например, везде родители в какой-то мере рассчитывают на любовь своих детей. Способ выражения этой любви может сильно зависеть культуры, в которой воспитаны дети, но само чувство любви безразлично к культуре.

Можно представить это и иначе: культура определяет, каким образом набор абстрактных принципов транслируется в повседневное поведение. У всех у нас есть набор почти инстинктивных типов «поведения по умолчанию», которое запрограммировано в нас сызмальства и представляет нормы, принятые в нашей локальной среде. Например, если мы живем в культуре, которая осуждает насилие, то при возникновении конфликта нашим поведением по умолчанию (согласно этой культурной норме) будет применение слов, а не кулаков, чтобы разрешить конфликт вербально. Однако если выясняется, что некто, угрожающий жизни другого человека, не желает прислушиваться к доводам, культура допускает насилие как последнее остающееся средство потенциальной жертвы (или сотрудника правоохранительных органов). Но это исключительный случай. Обычно за нарушение культурных норм, которое не было оправдано исключительными обстоятельствами, предусматриваются санкции, как формальные, так и неформальные.

Возможно, лучше всего выразился мой коллега Филипп Кратчен, когда написал следующее (в неопубликованной статье):

Нашим поведением управляют три силы:

- *Человеческая природа*: она передается нам по наследству и одинакова во всех человеческих культурах.
- *Культура*: общие программы поведения, которые приобретаются, а не наследуются.
- *Личность*: компонента, представляющая дополнительный уникальный набор программ, управляющих поведением, не являющийся общим с другими людьми. Он является частично унаследованным, частично приобретенным.

Если мы не сталкиваемся с другими культурами, нам трудно отличить культуру от человеческой природы. Естественно предположить, что все эти аспекты универсальны, но это не так. Важно уметь понять, когда характеристики обусловлены культурой, т. е. это общие черты группы индивидов, а не индивидуальные особенности. Отчасти легче оправдать того, кто не стал преодолевать культурный барьер, чем простить то, что воспринимается нами как личная аномальность.

Сильные и слабые культуры

В продолжение той же линии я полагаю, что есть два типа культуры: сильный и слабый. Они характеризуют то, каким образом культура транслирует свои принципы или ценности в повседневную жизнь.

В сильной культуре абстрактные принципы (ценности) транслируются в повседневную жизнь людей весьма непосредственным образом. Например, у военных есть определенный набор ценностей и очень сильная культура. Даже если вы не согласны с этими ценностями, вам придется согласиться, что в повседневной жизни они воплощаются очень строго и последовательно; они осуществляются посредством внешних законов и правил, а также обучения, направленного на внутреннее усвоение. Превалировавшая в шестидесятые годы XX века культура также была сильной и характеризовалась недоверием к власти и желанием оспорить все социальные условия. Опять же, согласны вы с этими ценностями или нет, приходится признать их мощное влияние на общественное поведение в то время. Я, например, помню, как трудно было «организовать» демонстрацию за мир, поскольку участники были настроены столь антиавторитарно, что лидерам приходилось объяснять и оправдывать каждую свою «просьбу»,¹ а «принуждение» исходило почти исключительно со стороны своих же товарищей.

Не все культуры настолько сильны. В некоторых есть набор общепринятых абстрактных ценностей, но они не наполняют повседневную жизнь. Например, в западной культуре религиозные ценности занимают очень разное место в повседневной жизни тех, кто ходит в церковь. Иногда церкви и конфессии обладают очень строгой культурой и стремятся внедрить религиозные догматы во все поступки и помыслы денно и нощно. Другие конфессии, напротив, *снисходительно* относятся к регулированию по-

¹ Однако любопытно, что у некоторых членов этой культуры были «не подлежащие обсуждению» требования. Мы жили в интересное время.

вседневной жизни и рассматривают в качестве наивысшей ценности саму веру.

Сила культуры зависит в конечном счете от двух факторов. Один из них – это степень кодификации ценностей культуры и эффективности донесения их до каждого. Второй – это тяжесть наказания, которому подвергаются люди за нарушение культурных норм. Очевидно, самые сильные культуры те, в которых все члены отчетливо понимают и знают «кодекс», а также осознают строгость наказания за его нарушение. В то время как сильные культуры успешно переводят свои абстрактные принципы в повседневные действия, слабые культуры делают это значительно менее предсказуемым и эффективным образом.

Определение корпоративных ценностей

К сильным и слабым культурам мы еще вернемся, но сначала я хочу поговорить о том, как мы определяем ценности.

Иногда мы говорим о *базисных* ценностях, как если бы именно эти ценности превосходили другие, менее достойные, ценности. Но потом я с удивлением читаю, что у некой компании есть 17 базисных ценностей. По-моему, это бессмыслица. Едва ли в «базе» может найтись столько свободного места.

И все же я считаю, что одни ценности выше других, и мы должны выбрать те, которыми будем руководствоваться в своей жизни. Возможно, кто-то способен хранить в своей голове и в сердце некую иерархию ценностей, но для меня это слишком сложно. У меня есть очень ограниченный набор ценностей, которого я стараюсь придерживаться сам и согласен которому оцениваю других. Благодаря тому, что он ограниченный, мне не приходится заботиться о порядке и приоритетности, а также уравнивать эти ценности между собой.

Я также пришел к выводу, что нет смысла считать «ценностями» те качества, против которых и так никто не возражает. Возьмем, например, «гражданские обязанности»: кто скажет, что мы *не должны* быть хорошими гражданами? Неприятность, однако, в том, что это слово всегда толкуется в соответствии с «местными» представлениями о том, каким должно быть поведение хорошего гражданина в данной культуре. Парадокс состоит в том, что чем более общим и туманным является термин, чем ближе он к тому, чтобы стать «универсальной» ценностью, тем более узкой и культурно-

обусловленной становится его интерпретация.¹ Ценность такой ценности оказывается весьма относительной.

Следовательно, определяя набор желаемых ценностей для корпоративной культуры, я бы выбрал термины, исходя *не* из их относительной распространенности, а из практичности для описания такой компании, которая стоит того, чтобы в ней работать. Я хочу работать с людьми, которые разделяют эти ценности. Если они не нравятся кому-то, то я надеюсь, что он найдет себе другую организацию, ценности которой подходят ему больше. В конечном итоге это принесет больше радости и мне, и ему.

В моем списке три пункта – достаточно мало, чтобы постоянно хранить их в «регистрах» памяти.² Это честность, сосредоточенность на клиенте и результаты (integrity, customer focus, results).

Честность

Будучи краеугольным камнем всего благородного, *честность* требует от нас вести себя прямо, правдиво и открыто во всех наших отношениях – внутренних и внешних, с коллегами и клиентами. Пусть в мире существуют нечестные люди, но мы не должны допускать их в свою компанию. Мы можем позволить себе сотрудничать только с теми, чье поведение соответствует нашим стандартам.

Честность – это высокое требование, но крайне просто узнать, действовали ли вы честно. Это так же просто, как отличить добро от зла. Это не сложно.

Многие желательные характеристики корпоративной культуры, о которых я писал в предыдущих главах,³ основаны на честности: атмосфера высокого доверия, ответственность за обязательства, отсутствие нездоровой политики, подлинно совместная работа и открытый обмен идеями. Без честности нельзя надеяться на достижение или создание чего-либо перечисленного. Ниже я поговорю о том, почему эти характеристики имеют такое важное значение в организации, ведущей разработку ПО.

¹ Иногда утверждают, что нет универсальных ценностей, подходящих всем и каждому, однако некоторые, особо мягкие, на это претендуют. Известно даже американское разговорное выражение для обозначения того, против чего никто не возражает, – «apple pie and motherhood» (яблочный пирог и материнство).

² В прежние времена программисты могли указывать объекты, которые нужно хранить в памяти, доступ к которой происходит особенно быстро, на том основании, что они будут часто использоваться. Такие места назывались «регистрами».

³ См., например, главы 13, 14 и 15.

Сосредоточенность на клиенте

Каждому, кто занимается бизнесом, постоянно приходится терзаться необходимостью делать трудный выбор. Иногда выбор очевиден: обычно не возникает проблем, если надо сделать что-то явно необходимое или отказаться от очевидно неприемлемого. Однако когда требования противоречивы, границы нечетки или доводы не очень убедительны, все становится сложнее. В таких случаях успешность означает, что правильные решения принимаются чаще, чем если бы выбор основывался на простом результате подбрасывания монетки.

Я говорю здесь не о технических решениях. Имеются в виду обращения к здравому смыслу типа «выпускать продукт сейчас или поработать еще несколько недель и улучшить его?» Обычно на такие вопросы нет однозначного ответа. Но есть ли простой критерий, основываясь на котором можно выработать решение?

Я считаю, что существует. Я всегда задаю вопрос: «Как будет лучше для клиента?»

Однако ситуация осложняется тем, что клиент редко бывает один.¹ Поэтому надо отобразить свою клиентуру в виде некоего распределения и попытаться определить, что окажется лучше для основной массы клиентов — наибольшее благо для наибольшего числа. Если вы способны поставить себя на место клиента и рассудить, что для него лучше, то чаще, я уверен, сумеете принять правильное решение.

Учтите, что ориентация на нужды клиента может иногда привести к существенным внутренним трудностям. Но если не делать этого, то вы просто отложите преодоление этих трудностей на более поздний срок. И, скорее всего, эти отложенные трудности будут значительно больше тех, которые вам пришлось бы перенести, если бы вы сразу попытались выработать ответственное и чуткое решение.

Ориентация на клиента имеет многочисленные проявления в культуре: стремление выпускать продукты в срок, увлеченность творчеством и качеством и выпуск продуктов, действительно удобных для работы. Но все начинается с того, что вы найдете в себе мужество задать вопрос: а что лучше для клиента?²

¹ И если даже он один, в его организации могут действовать противоречивые требования.

Результативность

Третья ценность относится к результатам. Я верю в результаты, а не в оправдания.

Насколько я вижу, сегодня масса времени и усилий тратится на оправдания. Все выглядит так, будто хорошее объяснение провала может служить допустимой заменой требуемому результату. Но это не то, на чем можно строить свою деятельность.

Проще всего определить, что хорошо, а что плохо, оценив результаты. Не намерения. Не усилия. Не «делает то, что от него требуют». Не «прост в работе». Все это не имеет значения, если не достигнуты результаты. Обратите, однако, внимание, что я не говорю «успех любой ценой» или «цель оправдывает средства». Подобные макиавеллизмы означают ваше нежелание действовать прямо, а это недопустимо.¹

Наконец, надо понимать, что большинство достойных целей достигается не в спринтерском забеге, а в марафонском. Важна настойчивость. Приветствую вас, те, для кого «провал – не вариант выбора»!²

А в применении к ПО...

Иногда меня обвиняют в самых широких обобщениях под маской об-суждения разработки ПО. Виновым себя не признаю! Просто прин-ципы, в которые я верю, универсальны, и их значение не ограничивается областью программных разработок. С другой стороны, готов продемонст-рировать, что указанные мной конкретные ценности *имеют отношение* к программным разработкам: при их несоблюдении программные про-дукты и компании деградируют.

Вернемся к честности – одной из наших базовых ценностей: мы нико-гда не лжем своим клиентам и никогда не лжем самим себе. Мы поставляем

² Естественно, это означает знание своих клиентов и их трудностей. Я не могу се-бе представить, как можно без этого заниматься каким бы то ни было бизнесом – не только программным.

¹ Это один из тех редких случаев, когда приходится расставить приоритеты даже среди нашего малого числа базовых ценностей. Вне всяких сомнений, чест-ность всегда главенствует над остальными двумя. Сосредоточенность на клиенте и ориентирование на результаты редко вступают в конфликт между собой, по-этому нет нужды распределять приоритеты между ними.

² Автором считается Джин Кранц из NASA. Сказано во время операции по спасе-нию Apollo 13.

высококачественный продукт в обмен на некоторую сумму денег, и клиент вправе рассчитывать на получение потребительской стоимости за свои деньги. В этом простом уравнении нет ничего специфического для программного обеспечения.

Тем не менее, чтобы производить программный продукт, которым клиенты смогут пользоваться на протяжении длительного периода времени, требуется честность. Во-первых, продукт должен поддерживаться на протяжении ряда последующих версий; нельзя просто залатать его для следующей версии, а потом забыть и выбросить. Для этого архитектура продукта должна обеспечивать возможность сопровождения и периодической модификацию в соответствии с вновь появившимися требованиями. Кроме того, должна быть обеспечена поддержка продукта, чтобы при возникновении проблем и обнаружении дефектов организация могла справиться с ними и предоставить клиентам соответствующие решения. Для поддержки и развития продукта на протяжении многих лет необходима приличествующая инфраструктура, а для соответствующих капиталовложений необходимо честно рассматривать долгосрочные перспективы. Компании с девизом «поставил и забыл» долго не живут.

Атмосфера высокого доверия и сотрудничества важна для производства любого крупного программного продукта. Почему? Ответ прост: ПО – это продукт, очень сложный по своей природе. Многочисленные отдельные участники команды ежедневно принимают тысячи решений, крупных или мелких, и все они могут оказать влияние на клиентов. Если бы мы приняли систему разрешений и контроля с целью гарантировать «корректность» всех таких решений, всякое движение вперед прекратилось бы. Вместо этого мы считаем, что каждый участник принимает верные и, как правило, самостоятельные решения. Конечно, важные и большие решения должны подвергаться контролю, но основная масса решений должна приниматься быстро и реализовываться действенно и умело. Это невозможно в отсутствие честности на всех уровнях организации.

Теперь поговорим подробнее о сосредоточенности на интересах клиента. Умники, вещающие о качестве, обожают забраться на трибуну и прочесть лекцию о необходимости повышать качество поставляемых нами программных продуктов. Но все несколько сложнее. Качество – весьма субъективная цель. Одни измеряют его количеством дефектов, другие говорят о юзабилити (для кого-то это «пригодность для работы», для кого-то – эффективность), у третьих есть совершенно другие критерии. Но ни в одном слу-

чае качество нельзя оценить абстрактно: всегда существует некий компромисс между качеством и какой-то комбинацией стоимости и времени.

Вот почему я считаю, что сосредоточенность на клиенте – это правильно выбранная ценность, и проявлением этой ценности в культуре оказывается то, что все члены организации заботятся о клиенте: разработчики, тестеры, составители документации, менеджеры продуктов, сотрудники службы поддержки и технической помощи, торговые агенты и специалисты по маркетингу – да, буквально все. Ответственность за работу с клиентами не ограничивается какой-то одной группой: это ответственность каждого. А атмосфера высокого доверия плюс сосредоточенность на интересах клиента – это мощное сочетание.

Что сказать о результатах? Напомню, что системы следует проектировать, кодировать, документировать, тестировать по многим параметрам, а затем собирать, упаковывать и поставлять. Если не уделять особого внимания результатам, то продукт просто никогда не покинет пределов организации. Слишком много есть всяких взаимозависимых вещей. Урок старой истории «потому что в кузнице не было гвоздя...»¹ в полной мере применим к разработке ПО. Большинство проектов запаздывают из-за накопления промежуточных задач, которые задерживаются, откладываются и просто не завершаются. Каждая из этих маленьких заминок не представляется важной, но их последовательное соединение оказывается губительным.

Обратной стороной медали является скрытая опасность стремления к «идеальным» результатам. Каждому должен быть ясен самый важный результат, который должен быть достигнут: своевременная поставка высококачественного программного продукта. Какой бы совершенной ни была спецификация требований к программному обеспечению, она не имеет значения, если отсутствует результат, видимый клиенту, – продукт.

Создание сильной культуры

К каким образом в сфере разработки ПО можно реализовать эти три ценности и построить на их основе сильную культуру?

¹ История начинается с того, что из-за отсутствия гвоздя не стало подковы, потом не стало лошади и т. д. вплоть до проигрыша сражения и потери царства, и все из-за того, что «в кузнице не было гвоздя».

Три обязанности руководства

Во-первых, *лидеры должны служить примером*. Если вы лидер, то в ваши обязанности входит укреплять и распространять культуру и ценности предприятия, начиная с честности. Наши лидеры должны соответствовать еще более высоким стандартам честности, чем все остальные. Я имею в виду всех наших лидеров, а не только тех, кто находится на самой вершине. Руководство составляет часть культуры компании, и независимо от вашего места в корпоративной иерархии положение лидера означает более высокую ответственность. Единственный способ продемонстрировать честность в качестве лидера – это действовать с честностью во всем.

Поразительно, что Конгресс Соединенных Штатов почувствовал необходимость принять закон,¹ требующий от руководителей американских компаний клясться в достоверности своих финансовых отчетов. Разве мы не можем просто полагаться на точность этих отчетов? «Подделка бухгалтерских документов» – моральный эквивалент изготовления фальшивых денег, а Данте неспроста поместил фальшивомонетчиков внизу восьмого круга ада. В его представлении ад состоял из девяти концентрических кругов, поэтому положение внизу восьмого круга близко ко дну преисподней.

Во-вторых, для создания культуры, в центре интересов которой находится клиент, лидеры должны *общаться с клиентами*, чтобы иметь ясное представление об их потребностях. Только тогда они могут задать правильное направление для всей организации.

Наконец, лидерство нужно для установления *зависимости оплаты от эффективности работы*. Менеджеры компаний, занимающихся разработкой ПО, должны уметь объективно оценивать эффективность труда и выплачивать людям вознаграждение за их вклад в работу организации и ни за что иное.²

Преимственность

Помимо удовлетворения этих трех потребностей нам необходима *преимственность*. Распространение культурных ценностей имеет смысл лишь тогда, когда оно продолжается несмотря на все изменения, случающиеся в деловом мире. Рассмотрим некоторые явления, угрожающие преимущественности в софтверных компаниях.

¹ Акт Сарбанеса-Оксли от 2002 г.

² См. главу 16.

Все тот же черт – расширение

Успешные компании неизбежно расширяются, как бы малы они ни были изначально. При этом они должны позаботиться о сохранении своей культуры, чтобы не утратить те самые ценности и традиции,¹ которые привели их к успеху. Но это не так-то просто.

Если компания расширяется не слишком быстро, тогда все не очень сложно. Есть время на тщательный подбор кадров, чтобы новые люди хорошо вписывались в культуру организации. Напротив, быстрый рост представляет собой двойную опасность. В поспешном стремлении заполнить вакансии² легко утратить дисциплину, место которой займет неряшливость. В крайнем случае мы набираем людей, которых не приняли бы, не будь такой спешки. Положение усугубляется тем, что мы не объясняем им, в какую культурную обстановку они попадут, потому что спешим набрать следующую партию работников.

Обратите внимание: это не значит, что мы набираем людей с неправильными ценностями. Нет, обычно с расширением организации просто происходит общее ослабление культуры. Например, случаи, когда интересам клиента не уделяется первостепенное внимание, перестанут вызывать резкую критику. Со временем подтверждается справедливость закона Грешэма: плохие деньги вытесняют из обращения хорошие деньги.³ Живущие по старым канонам оказываются в меньшинстве, возбуждая сначала любопытство, а потом и насмешку остальных, и в конце концов всеми игнорируются.

¹ Традиция – это любой элемент культуры, который укоренился так, что люди даже не помнят, из какой ценности он возник.

² Одно из самых нелюбимых мной выражений в бизнесе.

³ Здесь требуются некоторые пояснения. Сэр Томас Грешэм, советник королевы Елизаветы I, заметил, что когда вводят новые деньги для замены старых, обесценившихся, люди склонны копить новые деньги и при этом тратить как можно больше старых. Отсюда и идет выражение «плохие деньги вытесняют хорошие». Однако в бизнес-контексте мы можем обратиться к этой старой поговорке, чтобы описать происходящее с маленькой компанией, обладающей сильной культурой, когда она превращается в большую компанию со слабой культурой. Более новая, слабая, распыленная и менее желательная культура оказывается свойственной большому числу людей, поэтому с течением времени более старая, сильная и ценная культура маргинализируется среди все меньшего числа приверженцев и в конечном итоге погибает. Так «тирания числа» одерживает, к сожалению, победу.

На смену сильной культуре часто приходит бюрократическая система, призванная контролировать и осуществлять широкий круг стратегий и процедур, которым несвойственны прежние представления о «правильных действиях». Если культура организации слаба, последняя скатывается к дотошному вниманию к мелочам, как будто можно восстановить дух закона за счет роста численности его букв. Лидеры считают, что все остальные начнут принимать неверные решения, и потому плодят многочисленные правила, которые должны направлять и корректировать их действия. Верным признаком установления господства бюрократии является замена десяти заповедей руководством в 247 страниц, которое собирает на полке пыль.

Слияния и приобретения

После расширения следующим по разрушительному воздействию на сильную культуру событием является слияние или приобретение. В программной индустрии периодически происходят укрупнения; многие компании, и в их числе Rational Software, многие годы расширяются естественным образом, а потом прибегают к слияниям или приобретениям для продолжения или ускорения роста. К несчастью, в софтверостроительных компаниях, по-видимому, гораздо хуже переносятся такие операции, чем в других областях производства. Возможно, это вызвано наличием у них широкого спектра культур и ценностей. Самой главной причиной неудач при объединении компаний, производящих ПО, оказывается несовместимость их культур. Даже при близости двух культур слияние может быть затруднено из-за массы технических причин. Кроме того, если две компании географически существенно отдалены одна от другой, их раздельное положение служит причиной изолированности. Каковы бы ни были главные причины, при наличии принципиальной несовместимости большинство слияний компаний в программной индустрии просто оказываются неудачными.¹

Однако это препятствие преодолимо. Когда наши прабабки с семьями пионеров пересекали прерии в фургонах, они везли с собой куски теста «на разводку», чтобы печь на новом месте такой же хлеб, к которому при-

¹ Уолл-стрит подтверждает это почти при каждом объявлении о слиянии софтверных компаний. Обычной реакцией является падение цен на акции обеих компаний перед совершением объединения. Только после того как слияние доказало свою успешность, цена на акции новой компании восстанавливается. Хотя то же явление наблюдается и при других слияниях (даже нетехнических компаний), для софтверных компаний оно выглядит особенно резким.

выкли у себя дома. Это тесто содержало дрожжевую культуру,¹ которая порождала новые хлеба. Каждый раз, выпекая хлеб, они сохраняли кусочек теста, чтобы потом смешать его с новым.

Аналогично, переводя одного или нескольких человек, предпочтительно старшего возраста, в новое место, софтверные компании обеспечивают «начальную закуску» для передачи культуры и ценностей в присоединившуюся или приобретенную компанию. Всякое слияние или приобретение, при которых пренебрегают передачей культурной информации на раннем этапе, чревато грядущей катастрофой.²

Один крупный клиент или партнер

Еще одна интересная угроза возникает, когда есть один крупный клиент или партнер с очень сильной культурой. В таком случае скрытые и не очень влияния могут проникнуть в вашу организацию. Например, принятый клиентом или партнером стиль сообщения о результатах может наложиться на некоторые проекты; постепенно это может распространиться среди сотрудников, и все внутренние подразделения начнут докладывать таким же образом.

Иногда это может пойти на пользу: всегда стоит поучиться у других. С другой стороны, мы должны стараться, чтобы клиент или партнер разделил наши ценности, а изменения касались культурного проявления этих ценностей, но не самих ценностей.

Наконец, печальная правда жизни в том, что партнерство в программном бизнесе затруднено. Сегодняшний партнер завтра может стать конкурентом, и наоборот. Произвести оценку возможных последствий представляется читателю – в качестве упражнения.

¹ Согласно словарям слово культура употребляется в двух смыслах – как «обычаи, учреждения и достижения определенной нации, народа или группы» и «выращивание растений, животных или производство клеток или тканей». Предоставляю вам поразмышлять, является ли такая лингвистическая двусмысленность совпадением или под ней лежит нечто большее.

² Поразительная ирония состоит в том, что когда в 2002 г. я писал эти строки, Rational Software сама была приобретена IBM. О покупке было объявлено почти одновременно с публикацией. До сих пор все шло весьма и весьма хорошо. Держу пальцы скрещенными.

Новые направления

Еще одной угрозой для сильной культуры является открытие нового направления, например новой линейки продуктов или филиала в заморской стране. Здесь принцип «начальной закваски» является решающим. Мне лично известна новая организация, построенная с нуля на другом конце света, которая тщательно воспроизвела «материнскую культуру» только потому, что ее создавал один человек, которому был досконально известен первоначальный рецепт. Он набирал и обучал каждую новую партию работников. Через десять лет медленного, но устойчивого роста эта организация стала одной из сильнейших в компании как в смысле культуры, так и в отношении продуктивности, несмотря на свою крайнюю географическую удаленность от головного офиса.

Особенно остерегайтесь попыток создания новых групп, которым явно ставится задача начать радикальную смену культуры: они почти всегда обречены на провал. И вот почему: если это такие важные изменения, засучите рукава и возьмите на себя труд провести их во всей организации. В противном случае новая организация будет отторгнута и вызовет возмущение в остальной части компании. В случае успеха возмущение станет еще сильнее. С ростом могущества и влияния возникает признак гражданской войны, и единственным решением может быть разделение. В итоге будут утрачены все выгоды для первичной организации. С другой стороны, если эксперимент окажется неудачным, это может быть следствием того, что отколовшаяся группа чувствовала себя покинутой (или действительно была таковой).

Если вы ищете работу...

Последний пункт моего плана – совет тем, кто ищет работу. Я глубоко убежден, что крупнейшим фактором, определяющим ваше удовлетворение и успех в любой компании, является ваше восприятие ее культуры и ценностей. Почти все остальные переменные в этом уравнении могут и должны со временем измениться: ваши функции, обязанности, непосредственный начальник, фирма и зарплата. Проблемы в любой из этих областей можно решать и решить с течением времени. Но если существует глубокое несоответствие между существующей культурой и вашими представлениями о правильной культуре и здоровых ценностях, вы будете бороться с тем, что ежедневно раздражает вас в хорошие времена и совершенно погубит в плохие. Помните, что культуры и ценности меняются крайне мед-

ленно. Больше шансов, что вы постепенно адаптируетесь к культуре, а не эта культура изменится в направлении ваших вкусов. Поэтому ищите совместимую культуру, если, конечно, вы не любитель плыть против течения.

Как узнать, каковы подлинная культура и ценности компании? Лучше всего поговорить с теми, кто работает в компании сейчас или работал в недавнее время.¹ Попросите их честно ответить, что им нравится, а что нет. А во время формального интервью в компании сделайте две вещи. Во-первых, попытайтесь определить, насколько важными считаются культура и ценности, по тому, зададут ли интервьюеры вопросы для проверки совместимости между вами и их организацией. Если ни одного подобного вопроса вам не зададут, будьте осторожны. Это означает, что либо они небрежно осуществляют набор, либо корпоративная культура крайне слаба.

Во-вторых, когда настанет ваша очередь задавать вопросы, постарайтесь разговорить их на темы культуры и ценностей. Не бойтесь поставить их в затруднительное положение; например, открыто поинтересуйтесь, какая ценность считается важнейшей в компании или какова отличительная черта наиболее успешного служащего компании. Если у них правильная культура и ценности, они поймут причину вашего интереса и воспримут ваши вопросы как признак вашей серьезной заинтересованности. Если рассказы большинства людей, с которыми вам удастся побеседовать, окажутся согласующимися между собой и связными, есть основания считать культуру сильной: «кодекс» известен всем и большинством воспринимается одинаково. После того как вы определите наличие сильной культуры, уже проще выяснить, согласуется ли она с вашими ценностями.

Если вас примут на работу, полезно сразу приступить к углубленному изучению культуры и ценностей вашей новой компании. Не стесняйтесь задавать вопросы. Чем быстрее вы сможете усвоить поведение, принятое в качестве нормы в вашей компании, тем меньше ошибок совершите и тем более продуктивно сможете работать. Этот совет применим и в том случае, если вы работаете в компании, которая сливается с другой или приобретена ею.

¹ Возможно, придется учесть некоторые обстоятельства. Некоторые нынешние работники могут быть излишне ревностны. Прежние работники могут оказаться ожесточенными без достаточных оснований. Однако непосредственно полученные данные всегда полезны.

Краткие итоги

Я считаю, что культура должна усиленно пропагандировать такие ценности, как честность, сосредоточенность на клиенте и конечные результаты. Разработка и поддержка программного обеспечения наиболее успешны, когда ведутся талантливыми, творческими и компетентными личностями, которые в совокупности действуют как дружная команда. Сохранение сильной культуры происходит благодаря личному примеру руководства и стараниям передать ценности в условиях роста, слияний и приобретений, давления со стороны крупных клиентов и партнеров и новых бизнес-начинаний. Так же, как образование – это то, что остается, когда забыто все остальное, культура и ценности – это то, что правит, когда никто не видит. О них стоит позаботиться.

Резюме

Трудно писать о культуре и ценностях и не начать читать проповедь. Например, если человек честен, нет смысла проповедовать ему честность, а если не честен, то вряд ли эта книга на него повлияет – так в чем же смысл? Я ставил себе задачу показать важность честности тем людям, которые хотели бы ее соблюдать, но часто бывают вынуждены хитрить. Не сдавайтесь, будьте упрямы, потому что это важно.

То же самое относится к сосредоточенности на интересах клиента и конечных результатах. Дело не в том, что «раньше все было лучше». И мы тогда тоже не понимали самых элементарных вещей! Сохранение преемственности и постоянства, несмотря на удаленность в пространстве и времени, требует приверженности небольшому числу основных принципов. Здесь я лишь попытался рассказать о тех из них, которые оказались полезными для меня.

Мы приближаемся к концу нашего путешествия. В заключительной главе я попытаюсь для пользы читателя связать все в единое целое.

ГЛАВА ДВАДЦАТЬ ЧЕТВЕРТАЯ

Соединяем все вместе

Поразительно наблюдать за тем, как меняются люди со временем. При этом одни растут и развиваются, а другие становятся вечными узниками образа действий, который мешает им достичь счастья и благосостояния. Кто-то достигает успеха в одних сферах жизни и терпит сокрушительное поражение в других. Некоторые добиваются исключительного успеха по общепринятым меркам, совершенно не испытывая при этом удовлетворения. Есть ли какая-то модель, обобщающая все такие наблюдения?

Долгие годы наблюдений и размышлений привели меня к убеждению, что легче в этом разобраться, если упростить задачу. В моей модели есть три базовых состояния, характеризующихся на идише словами *шлеппер*, *махер* и *мени* (*schlepper*, *macher*, *mensch*). Сначала я опишу эти состояния и то, как люди переходят из одного состояния в другое. Затем я объясню, как можно застрять в одном из предшествующих состояний и как бороться с такой ситуацией. Кроме того, я поговорю о том, что в различные периоды своей жизни люди в каждый данный момент могут пребывать в разных состояниях. Наконец, я обращусь к вопросу о том, как люди количественно распределяются по разным состояниям и как это учитывать, живя в реальном мире.

Здесь я хочу немного уточнить значение слов. Я называю эти три состояния «фазами», потому что считаю, что они могут естественным образом последовательно сменять одна другую, причем в каждом человеке. Эти фазы наступают по мере того, как люди взрослеют, приобретают опыт, примиряются с окружающим миром и учатся находить компромисс между

системой своих убеждений и требованиями повседневной жизни. К сожалению, иногда люди застревают на одной фазе и не двигаются дальше. Это побуждает рассматривать их как «класс» людей. Однако слово «класс» ассоциируется с таким количеством других понятий, в том числе социальных, что я постарался не употреблять его.

Почему это важно? Мы склонны считать, что жизнь – сложное явление, и вопросам взаимодействия общественных групп в различных контекстах: семейном, деловом, трудовом и т. д. – посвящена масса научных исследований. В большинстве своем они недоступны обычному человеку. Я пришел к убеждению, что данная очень простая модель в значительной мере объясняет явления, наблюдаемые в реальном мире, и позволяет делать определенные предсказания. Простая модель, которую людям легко понять и применить, и которая оказывается действенной в 80% случаев, полезнее, чем сложная и трудная для применения, которая действует в 95% случаев.

Шлеппер

Начнем с первого состояния. Люди на этой фазе развития в совокупности называются шлепперами. Слово происходит от глагола «schlep», на идише означающего «тащить». В разговорной речи он также означает «пронести что-либо», например, «пронести вещи через аэропорт». В обиходе слово «шлеппер» указывает на ленивого и неаккуратного человека, но я хочу применить его с другим оттенком значения. Для меня «шлеппер» – это человек, находящийся на первой стадии своего развития.

В буквальном смысле шлеппер – это носильщик. В старые добрые времена идеальный пример шлеппера являл кэдди – мальчик, таскавший сумку с клюшками для гольфа. Когда что-то тащишь, не требуется особого напряжения мысли: ты выполняешь полезную, но преимущественно «черную» работу, обычно обслуживая кого-то другого. Быть шлеппером не слишком почетно, но не стоит и недооценивать важность этой работы.

Во-первых, если ты шлеппер, никто не запрещает тебе думать. На самом деле верно как раз обратное: поскольку работа шлеппера не требует особых раздумий, во время нее можно думать и учиться. Когда что-то волочишь, часто рождаются творческие мысли. Например, как бы сделать, чтобы тратить на эту работу меньше сил? В результате в каменном веке какой-то шлеппер изобрел колесо. Нежелание рутинно совершать скучную и неинтересную работу или тратить непомерные усилия для достижения обыденной цели часто приводит к возникновению идей даже у самого тупого

шлеппера – нужда (ощущаемая через боль или усталость) становится матерью прогресса. Мой опыт говорит, что люди, которым приходилось быть шлепперами, часто обнаруживают новые и интересные способы избежать нудной работы, даже если это касается новой для них области. У них развивается инстинктивная способность предвидеть возникновение тягостной работы и предотвращать это. Из экс-шлепперов, например, получают замечательные инженеры.

Вообще говоря, каждому следует пройти через шлепперство. Оно воспитывает характер, как бы банально это ни звучало. Оно учит нас смирению – тому смирению, которое говорит: «Если я что-нибудь не придумаю, то буду тащить лямку всю свою жизнь». У этого явления есть интересные стороны.

Шлепперы быстро начинают ощущать несправедливость жизни. Вот он я – молодой, умный, красивый и т. д. – и я должен ишачить на старого толстого идиота, который почему-то оказался моим хозяином! Как *это* получилось?

Иногда этот хозяин может оказаться действительно глупым – настолько, чтобы заставлять тебя ишачить больше, чем это необходимо. Иногда он может еще сильнее отравить тебе жизнь, намеренно проявляя жестокость. И поскольку тебе предназначено быть шлеппером, у тебя есть две возможности: продолжать молча «шлепать» здесь или отправляться «шлепать» в другое место. Третий вариант – поднять шум – обычно оказывается контрпродуктивным, поскольку шлепперы взаимозаменяемы по определению, и тем, кто много шумит, быстро находит замену.

Наблюдательный шлеппер обнаруживает некоторые поразительные вещи. Например, у молчаливого шлеппера развивается эрозия стенки желудка, и если он способен делать выводы, то постарается изменить свою работу или социальное положение, чтобы уменьшить переживания. Уходить, чтобы продолжать тянуть лямку в другом месте (вариант номер два), обычно смысла не имеет, т. к. это не решает проблему, потому что как все шлепперы взаимозаменяемы, так и работа для них всюду в принципе одинакова. Чаще всего получается переход «из огня да в полымя».

Пропускать фазу шлепперства опасно, даже если такая возможность предоставляется. Правда, кое-кому это удастся, например тем, кто родился в богатой семье. Им никогда не удастся испытать преимущества «шлепперства», например радость творчества, когда тянешь лямку, или гордость, испытываемую, когда удачно что-то дотащишь. В результате им оказывается неведомо, как живет большинство в этом мире. Слишком многое они счи-

тают само собой разумеющимся и плохо понимают реальность. А если вы не начали жизнь как шлеппер, то практически невозможно, чтобы вы когда-нибудь им стали.

Но что существеннее всего, вы не получите важных уроков – смирения, цены заработанного тяжким трудом доллара, органической несправедливости мира и напряженности жизни рядовых людей. Другой незаменимый урок дает общение с огромным разнообразием людей, с которыми встречается шлеппер в реальном мире – ворами, лжецами, мошенниками и теми, кого в менее политкорректные времена называли «простым народом». А самое главное, встречаются удивительные люди, которые разглядят в тебе нечто и подумают: «А что тут делает этот парень? Он явно способен на большее», – и что-то сделают для тебя. Они становятся нашими наставниками, тренерами и защитниками, и это один из путей покинуть фазу шлеппера.

Рано или поздно каждый шлеппер должен понять, что его прогресс зависит от способности выйти из фазы шлепперства. Но для этого требуются усилия. Можно остаться шлеппером навсегда и винить в этом пороки классовой системы, рыночный капитализм или что-то иное – все с одинаковой пользой. Выйти из этой фазы можно, если принять такие меры, после которых твой начальник скажет: «Послушай, я плачу тебе не за то, чтобы ты сам таскал мешки! Поищи другого, кто этим займется!» Часто для этого требуется получить образование, что-то придумать и просто сделать нечто, что покажет тебя в положительном свете. Необходимы, говоря словами Черчилля, кровь, пот, слезы и труд.¹ Вы должны показать, что можете приносить пользу на следующем уровне. Этот план состоит из двух частей. Во-первых, вы должны получить образование, приобрести мастерство, добиться результата, сделать дело. Затем кто-то, кто обладает влиянием, должен обратить внимание, что произошли перемены и вы готовы перейти из шлепперов в следующую фазу. Это те наставники, о которых я говорил выше.

Итак, все мы начинаем как шлепперы. В любой семье дети – это шлепперы. Можно рассматривать шлепперство как ученичество. Дети учатся на взрослого. Если они будут внимательны, не погибнут и иногда будут прислушиваться, то благополучно закончат учебу и перейдут во взрослые. Иначе они останутся детьми навсегда.

¹ Если вы хотите оставить следы на песке времен, советую надеть рабочую обувь. Вот более точная цитата Черчилля из его речи 13 мая 1940 года: «Я могу предложить вам только кровь, труд, слезы и пот». Впоследствии неправильное цитирование устоялось в языке. Вспомните название рок-группы «Кровь, пот и слезы».

Смиритесь с тем, что в любом занятии – на каждой новой работе, в каждом новом виде спорта, в каждой новой связи – вы начинаете как шлеппер. Как долго вы им останетесь, зависит от вас. И помните о необходимости соблюдать достоинство, пока вы находитесь в учениках.

Махер

Вторая фаза жизни – это фаза махера. Я полагаю, что происхождение слова связано с глаголом, означающим «делать». Эта вторая фаза самая длинная и в некотором смысле самая приятная фаза жизни. *Махер* – тот, благодаря кому совершаются вещи и происходят события, кто добивается результатов. Эта фаза чрезвычайно продуктивна, и большинство махеров получает глубокое удовлетворение от того, чем занимается. Некоторым махерам это так нравится, что они остаются в этом качестве навсегда, и это не так уж плохо. Если бы не махеры, мы бы все оставались шлепперами.

Махеры не просто изобретатели, предприниматели, художники и гении, хотя все эти люди обычно махеры. Махер выделяется тем, что создает новую ценность и изменяет порядок вещей. Обычно махер ассоциируется с высокой, не рядовой, продуктивностью. Тот, кто отрабатывает свои восемь часов и редко задерживается, не махер; это в некотором смысле лишь продвинутый шлеппер. Нет, махер принадлежит к той категории людей, про которых часто говорят: «Чтобы справиться с этим, нам нужен настоящий махер!». Во многих фирмах махеры – это «чудотворцы», те, на ком держится бизнес. Лакмусовая бумажка для проверки: уберите махера, и организация не просто сильно пострадает, она просто станет другой.

У махеров есть следующие интересные свойства:

- Обычно они очень сосредоточены, на грани одержимости.
- Они впечатлительны.
- Они ориентированы на результат.
- Они видят цель и могут поймать ее в прицел. Лучше не вставать на пути между махером и результатом, к которому он стремится.
- Махеры харизматичны – в хорошем и плохом смысле. Отсутствие у махера харизмы встречается редко, потому что эта черта очень часто связана с лидерством. Существуют исключения, но достаточно редко, чтобы останавливаться здесь на них.

Есть и отрицательная сторона. Махеры часто склонны думать, что цель оправдывает средства. Они бывают совершенно безжалостны. Те, кто сам

щепетил в отношении чувств других людей, часто прибегают к помощи махеров, у которых не возникает угрызений совести. Махер несколько не заблуждается в отношении того, за что ему платят деньги: за полученный результат. Но если говорить правду, то махер почти всегда готов работать бесплатно: успех – это очень сильный наркотик.

Махеры умеют сдерживать себя. Самые способные из них быстро обнаруживают, что опасно бить слишком много посуды. Настрой против себя слишком много людей, и ты не сможешь найти себе помощников, даже среди таких же махеров! Есть очень много противных молодых махеров, но очень мало противных пожилых махеров. Причина очевидна: махерам трудно продвигаться вперед, если они не могут сколачивать группы, состоящие из других махеров. Объем проблем, которые им предлагаются для решения, растет и достигает такой степени, что единственным вариантом становится игра в команде. Если махер не в состоянии выработать в себе искусство общения, необходимое для привлечения в игру других игроков, он окажется в изоляции, и его обгонят более толковые махеры.

Махеры получают некоторое побочное преимущество, которое не стоит недооценивать. В какой-то мере являясь «примадоннами», они могут устанавливать свои правила. Почему? Потому что многие люди и организации терпят довольно вызывающее поведение, если проблема, которую нужно решить, серьезна или ожидаемая прибыль достаточно высока. Поэтому махер может в значительной мере избежать мелочной тирании организаций и бюрократий, прямо поместив себя за рамками обычаев системы. Многие махеры идут этим путем просто потому, что иначе не могут функционировать: им необходим контекст, в котором они могут делать дело по своим правилам. В любом другом контексте они не справятся с задачей, потому что им придется подчиняться ограничениям, которые они считают для себя слишком обременительными. Но кому много дано, с того и спрос больше. Когда махер терпит неудачу, недостатка в желающих его закопать не бывает – с годами врагов у него становится только больше, и у них долгая память. Чтобы выжить, поставив себя вне системы, необходимо обладать действительно выдающимися качествами. Иначе твоя первая ошибка станет и последней.

Иногда махеров пьянит сила, которой они обладают, и они действительно выходят из-под контроля. В конечном итоге чрезмерно агрессивный махер обречен на саморазрушение, но сначала он вызовет заметный переполох. Махеры редко уходят тихо; обычно это происходит при ослепительном фейерверке. Высокомерие и здесь вмешивается, и поскольку

у махеров все происходит с размахом, их провал также обычно оказывается зрелищным.

Можно ли стать махером, не пройдя фазы шлеппера? Да, но это удастся редко. Махеры, которые в той или иной мере не прошли ученичества, обычно оказываются в чем-то неполноценными. Тяжело быть махером, если у вас нет крепкого понимания реальности, которое приходит в результате выполнения черной работы.

Махеры склонны оставаться в своей фазе, потому что они представляют собой элиту. В мире бизнеса они получают массу более или менее ощутимых наград в обмен на результаты, которые получает от них организация. Их постоянно привлекают к решению все более крупных и интересных задач. Это замечательная жизнь, которая предполагает невысокий риск – иногда отрицательная реакция организации, иногда преждевременные сердечные заболевания в силу выраженного поведения типа А. В большинстве своем махеры с этим справляются.

В других сферах жизни быть махером – значит быть компетентным, фактически это означает высочайшую компетентность в работе. Обычно махер рассчитывает добиться того же в любой сфере жизни. Большое разочарование он может испытать, если окажется, что его компетентность распространяется только на одну предметную область. Ergo, многие махеры становятся узкими специалистами, направляя всю свою энергию в ту область в которой они сильнее всего. Поскольку по своей природе они склонны к соперничеству, здесь есть реальная опасность застоя. Если ты лучше большинства своих коллег, то к чему еще можно стремиться?

Как ни высоко положение махеров, существует еще более высокое состояние. Слово из идиша, которым оно обозначается – *mensch* – практически непереводаемо.

Менш

Менш – это джентльмен, «утонченная натура». Но этим не вполне передается чувство, заложенное в фразе: «это настоящий менш!» Сущность понятия *менш* в том, что это человек с всеобъемлющим видением, склонный к интроспекции и философии и обладающий добротой. Менш умеет выслушивать и очень хорошо понимает чужую точку зрения.

Следует отметить, что слово *Mensch* в немецком языке означает, во-первых, человека и, во-вторых, мужчину. Хотя в таком смысле эта фаза стано-

вится доступной обоим полам, он более расширителен, чем в идише. На идише не все люди «менши».

Между «махерами» и «меншами» есть большая разница. Во-первых, махеры обычно резче в общении; менши добродушнее, мягче и спокойнее. У махеров есть чувство необходимости; у меншей есть чувство неизбежности. Менш серьезно полагает, что все должно уладиться. Шлеппера часто считают бестолковым или глупым, хотя вся его вина может заключаться в невежестве; махера считают толковым или умным; менша всегда считают мудрым. Вы обращаетесь к махеру, когда надо решить задачу сегодняшнего дня; вы обращаетесь к меншу, когда требуется перспективное решение. В каком-то смысле шлеппер не умеет делать ничего, махер – преимущественно тактик, а менш – стратег.

Прежде чем вы придете к выводу, что менш – это воплощенный Йода, должен отметить, что менш не просто источник рекомендаций, но и человек дела. Особое положение менша заключается в том, что он не только знает, что следует делать, но и действует соответственно, даже в ущерб себе. В отличие от махера, менш несколько не заинтересован в похвале за получение результата. Его интересует результат ради него самого и его редко волнует, знают ли люди о его участии. Типичным для менша поступком будет, например, крупное анонимное пожертвование.

Иногда из махеров получаются хорошие наставники, но практически всегда это побочный эффект при достижении их главной цели – результата. Махеры чаще обучают более молодых махеров, а не шлепперов. Напротив, из меншей получаются прекрасные тренеры и наставники, потому что они очень хорошо чувствуют потребности других; они всем помогают, потому что к каждому испытывают симпатию. Они также обладают способностью предвидеть существенные черты отдаленного будущего, поэтому понимают значение развития каждого и создания инфраструктуры. Они находят прелесть в том, чтобы привнести в систему энергию, не зная о времени и месте появления положительных последствий этого действия, но не сомневаясь, что они обязательно будут.

Менш также способствует погашению конфликтов в любой организации. Он выше драки, будучи преданным организации и ее целям, но не преследуя личных целей, в отличие от махера, у которого они всегда имеются. У махера есть своя территория, менш экстерриториален. Менш постарается стать миротворцем, посредником и изобретательно искать решение, когда кажется, что его не существует. Вопреки внешней видимости

менш оказывается очень эффективен. Его сила основана на способности дружно работать со всеми и на уважении, которое все испытывают к нему.

Можно ли стать меншем, не пройдя стадии махера? Есть две точки зрения.

Первая заключается в том, что переход от шлеппера к меншу равнозначен переходу от ученика к мастеру без промежуточной стадии просто квалифицированного работника. Согласно этой точке зрения мудрость, проявляемая меншем, накапливается за годы бытности махером. Хорошего махера выдерживают годами, и в итоге он становится меншем.

Недостаток этой точки зрения заключается в наличии явных исключений. Наряду с тем, что, как было отмечено, многие махеры никогда не переходят в менши, известно также, что есть люди с качествами меншей, которые никогда не были махерами. Они долгое время занимались черной работой, но не ожесточились при этом, а наоборот, набрались мудрости. Они добрые и надежные, полностью понимают людей и человеческие драмы и не скупаются на сочувствие. Здравость их суждений безупречна. Загадка лишь в том, откуда у них взялась мудрость.

Еще о меншах

Швейцарский физик и эколог Оливье Гизан 30 лет назад сказал мне, что ключ к взрослению в том, чтобы, держа глаза открытыми, не огрубеть сердцем. Важно в процессе созревания, когда мы учимся справляться с подчас отвратительными реалиями жизни, не стать циником. Типичный шлеппер – пессимист, а махер – циник. Менш является оптимистом. Он верит в то, что люди добры, а цивилизация сможет найти решение сложных проблем. Конечно, отчасти это связано с его собственной гуманностью, но он этого не замечает.

Уже упомянутый психолог Михай Чиксентмихайи описал в своей книге «Поток: психология оптимального переживания»¹ некую модель. По его теории существует конфликт знаний и навыков мастерства с решаемой задачей. Если задача слишком проста, людям становится скучно, они несчастны. Если задача очень сложна относительно компетентности, то люди напрягаются, испытывая тревогу. В случае разумного компромисса –

¹ Mihaly Csikszentmihalyi, Flow: The Psychology of Optimal Experience. (New York: HarperCollins, 1991) Чиксентмихайи М. «Поток: психология оптимального переживания» (Смысл, 2006). Подробнее о потоке см. главу 16.

не слишком легко и не слишком тяжело – достигается состояние «потока». Чиксентмихайи называет достижение состояния потока каналом потока, потому что при этом охватывается некоторый диапазон компетентности и сложности задач. Поток – это благодатное состояние, в котором велики достижения и испытывается чувство исключительной удовлетворенности; спортсмены называют это «нахождением в зоне».

Интересно, что в этой модели шлепперы будут несчастливы, поскольку постоянно будут ниже канала потока, работая над задачами, которые кажутся им скучными. Махеры, что очевидно, окажутся в тревоге, потому что чаще всего будут находиться выше канала потока: для них типично «превосходить задачи». А вот менши, я думаю, будут счастливы и плодотворны, потому что часто попадают в канал потока. Если главное – поймать поток, то менши, похоже, это поняли.

Любопытно, что меншу не обязательно быть старым, хотя многие из черт, ассоциируемых с меншами, приходят с возрастом. На самом деле менш – это состояние ума, которое доступно каждому при надлежащих взглядах и позиции.

Менши – счастливые люди. Их окружают счастливые люди. Они могут справиться с самыми неприятными неожиданностями в жизни и помочь другим сделать это. Их жизнь очень целостна и уравновешена, и душа их в мире.

Распределение людей по группам

На каждые 100 шлепперов приходится 10 махеров и один менш. Почему шлепперы так многочисленны? Простой ответ можно взять у Линкольна, который сказал бы: «Бог их любит, поэтому создал так много». И все-таки кажется, что неудовлетворенность должна была бы рано или поздно почти всех подтолкнуть к следующей фазе. Увы, это не так. Во-первых, большую роль играет лень: многие просто не желают делать то, что позволило бы им подняться. Во-вторых, требуется зрелость: для перехода в следующую фазу надо изменить отношение к жизни и взять на себя ответственность за свою судьбу. Проще жаловаться на систему и невозможность продвижения в ней, чем взять дело в свои собственные руки и добиться успеха, несмотря на препятствия. Наконец, чтобы продолжить свой рост, требуется совершить усилие. Выход из зоны шлепперов – фундаментальная перемена, которой многие боятся, потому что новый образ жизни, оказывается, лишен простых радостей, доступных шлепперу. Поскольку

ничто не дается без труда, многие шлепперы так и не решаются преодолеть эмоциональный барьер, что необходимо для перехода в следующую фазу. Полагаю, что три эти фактора – необходимость жертвовать, зрелость и фундаментальные перемены в жизни – объясняют, почему шлепперов так много.

Все это помещено в контекст реального и иногда жестокого внешнего мира. Мой опыт говорит, что интеллект и талант играют значительно меньшую роль в продвижении, чем напряженный труд и решительная позиция. В современной глобальной экономике, как мне кажется, открываются большие возможности и нет *непреодолимых* культурных, социальных или иных барьеров. Если вы уверите себя в том, что всем управляют внешние факторы, то будете обречены на роль шлеппера. Можно осилить тех, кто стоит на вашем пути, но никто не перенесет вас через препятствие, которое вы воздвигли для себя сами. На каждые 10 махеров приходится 100 шлепперов, и это представляется мне невероятной растратой человеческого капитала. Такое положение кажется мне все более неприемлемым по мере нашего дальнейшего движения в сторону информационной экономики. Рабочих мест для шлепперов становится все меньше, но то отношение к жизни, которое позволило им существовать так долго, остается.

Те, кто переходит в следующую фазу, проводят в шлепперах относительно небольшой период своей жизни. В категории махеров они проводят большую часть жизни, поэтому стоит постараться быть хорошим махером. Если бы махеры рассматривали этот период своей жизни как ученичество перед вступлением в менши, всем нам было бы немного лучше. Я не думаю, что они сильно уменьшили бы свою эффективность, а в конечном итоге все мы стали бы жить дольше и счастливее. Но изменить поведение махера трудно, потому что он считает, что его эффективность основана на тех качествах, которые отличают его от менша. Проблема именно в этом.

Я оцениваю соотношение меншей к махерам как один к десяти, но опасаясь, что эта оценка завышена. Меншей должно быть больше, потому что их нехватка ощущается постоянно. Слишком часто они живут в этом качестве очень недолго, и их дух оказывается долговечнее тела, которое его содержит.

Некоторые завершающие мысли по поводу модели

В модели сделаны определенные допущения. Вы начинаете как шлеппер, вырастаете в махера и надеетесь стать меншем. Это обычный

путь, исключения из которого отмечались на протяжении всей главы. Эта модель проста, но она несовершенна: когда имеешь дело с людьми и делаешь обобщения, встречаешься с «неприятными» исключениями.

Я знаю, что надо делать, чтобы стать махером, но дать рецепт, как стать меншем, я не могу. Меншем нельзя стать путем напряженного труда, как можно стать махером. Возможно, меншами рождаются, а не становятся. Спрашивать, как стать меншем, – это все равно что спрашивать, как стать мудрым, или как стать просвещенным.

Помочь может влияние, оказанное одним или двумя меншами, особенно в ранний период жизни, когда они могут служить образцом. Расти в детстве с отцом-махером и дедом-меншем может быть очень поучительно, если ребенок-шлеппер достаточно восприимчив.

Еще очень важно понимать, что всем меншам ничего не надо в обмен на их доброту, кроме того, чтобы вы передали ее следующему поколению.

Но разве я что-то знаю?

Резюме

Эта глава посвящена Рослин Розенталь Мараско (Roslyn Rosenthal Marasco, 1921–1998).

Менеджерам должна быть ясна последовательность (Д→И→З→М):

Данные → Информация → Знания → Мудрость

Стрелки означают «может быть источником для». Неправильное употребление на любом шаге приводит к ошибочному результату. Начать с «хороших данных» обязательно, но это не гарантирует появление чего-то полезного на другом конце.

Существует гигантский разрыв между первым этапом и четвертым. Слишком многие технологические компании заняты почти исключительно совершенствованием сбора данных, не развивая свои возможности на следующих трех этапах. Я полагаю, что такой стиль связан с менеджерами, имеющими техническую или научную подготовку: они предпочитают числа. Но числа бесполезны без их разумной интерпретации; в лучшем случае они помогут вам наполовину. Такой подход не приведет к существенным улучшениям в вашей организации, потому что плохие ответы, полученные на основании четких данных, принесут мало пользы. *Важно то, что вы сделаете с этими данными.*

Данные можно собирать автоматически. Переход ко второй стадии требует интеллектуального анализа и синтеза. Для достижения третьей стадии необходима интеллектуальная среда, из которой можно получить общие принципы. Наконец, переход на четвертую стадию требует опять-таки человеческого интеллекта и большого здравого смысла, чтобы обеспечить эффективное применение к реальному миру.

Осведомленность об окружающем мире фигурирует на разных уровнях абстракции. Чем выше уровень абстракции, тем более общим может быть применение нашей осведомленности к реальным ситуациям. Д→И→З→М соответствует возрастающим уровням абстракции, и абстрагирование правильных объектов становится более затруднительным с каждым следующим этапом. Данные конкретны и специфичны; мудрость абстрактна и универсальна; информация и знания представляют промежуточные состояния. Важно не смешивать информацию со знаниями, а знания с мудростью. Если в вашем распоряжении есть идея, полученная из данных, задайте себе вопрос, что это – информация, знания или мудрость. Ответ на этот вопрос часто может оказаться решающим.

Людей, находящихся в трех фазах жизни, можно представить себе как умелых агентов, осуществляющих различные преобразования. Нашу цепочку можно модифицировать, показав более характерные стрелки:

Данные →

(собираются и преобразуются *шлеттерами* в) →

информацию →

(которая преобразуется *махерами* в) →

знания →

(которые преобразуются *мениами* в) →

мудрость

Эта схема объясняет, почему мы купаемся в данных, извлекаем из них некоторую информацию, перегоняем часть ее в тяжело добытые знания и извлекаем из них крупинцы мудрости в качестве конечного продукта. У нас много чего есть, когда мы подходим к началу цепочки, и очень мало остается на другом ее конце. Сам процесс требует, чтобы, поднимаясь по ступенькам абстракции, мы формулировали свои выводы с помощью все меньшего числа более простых и мощных идей. Кроме того, количество доступных агентов сокращается в 10 раз на каждой следующей стадии: работа усложняется, а компетентных людей для ее выполнения все меньше и меньше на каждом уровне.

Агенты, которые выполняют работу на каждой стадии, применяют свой специфический набор профессиональных приемов: пусть у вас будет в 10 раз больше агентов, но если это «не те» агенты, то они будут бесполезны. Это классическая проблема согласованности импедансов: есть «нечто» в определенном состоянии, что должно быть преобразовано в «нечто» в следующем, более высоком, состоянии. Только люди с необходимым опытом и развитием могут осуществить необходимое преобразование, причем сделать это правильно и эффективно. Проведение различия между качеством и заурядностью затрагивает мораль и требует страстности, интеллекта и здравого смысла.

Наша задача как менеджеров, руководящих менеджерами, состоит в том, чтобы обеспечить расстановку нужных людей в нужном месте и в нужное время. Только тогда можно надеяться воспользоваться всеми плодами перехода от данных к мудрости.

Алфавитный указатель

A

Archer, James E., 127

B

Bernstein, Dave, 96

Boehm, Barry, 163

Bond, Robert, 122

Booch, Grady, 164, 191

Brooks, Frederick, 150, 168

C

Csikszentmihalyi, Mihaly, 241, 365

D

Dobbs' Journal, журнал доктора Доббса,
177

F

Feynman, Richard, 49

Field of Dreams (Поле грёз), фильм, 123

FORTH, язык программирования, 118

G

GIGO (Garbage In, Garbage Out), мусор
на входе – мусор на выходе, 51

Guisan, Olivier, 365

H

Hamming, Richard, 54

Hello, world, программа, 113

Heppeneimer, T. A., 52

I

ICD (Implanted Cardioverter Defibrilla-
tor), имплантируемый кардиодефиб-
риллятор, 31

K

K&R, 113

Kennedy, John, 41

Kernighan, Brian W. & Ritchie, Dennis M.,
113

Kruchten, Philippe, 82, 191, 342

KSLOC (kilo source lines of code), тысяча
строк кода), 164

L

Larsen, Howard, 241

Love, Tom, 261

M

Marasco, Andrew, 52

Marasco, Roslyn Rosenthal, 368

Meretsky, Wayne, 77

Moore, Geoffrey A., 184

P

Perrow, Mike, 202

R

Rational Software, 34, 81, 108

Rational Unified Process, 190

Royce, Walker, 82, 163, 191

S

Sadler, Mark, 282

Standish Group

отчет, 149

Standish Group, отчет CHAOS, 146

S-кривые, 182

U

UCM (Unified Change Management), унифицированное управление изменениями, 128

UML (Unified Modeling Language), унифицированный язык моделирования, 103

Unified Change Management (UCM), унифицированное управление изменениями, 128

уровень абстракции, 109

Urmí, Jaak, 261

W

Walker, John, 92

Westheimer, F. H., 92

Wideman, Max, 155

A

альпинизм

экспедиция как метафора проекта разработки программного обеспечения, 56

анalogии, некорректные, 266

Арчер, Джеймс *см.* Archer, James E.

атмосфера высокого доверия, 210, 348

аудитория данной книги, 35

B

бартер

в политике, 207

Бернштейн, Дэвид *см.* Bernstein, Dave

Бонд, Роберт *см.* Bond, Robert

Бозм, Барри *см.* Boehm, Barry

Брукс, Фредерик *см.* Brooks, Frederick

Буч, Гради *см.* Booch, Grady

B

«Ваза», гордость королевского флота Швеция, 261

ведение переговоров, 214

вероятность успеха проекта, 140

Вестхаймер, Ф. Х. *см.* Westheimer, F. H.

вознаграждение

влияние на мотивацию, 244

выбор правильного, 245

на основе трудности задачи, 246

на основе уровня мастерства, 245

неправильное, крайние ситуации, 249

ограниченность возможностей манипулирования, 255

функциональная зависимость от уровня квалификации и сложности задания, 258

вознаграждение профессиональных программистов, 240

второй закон Ньютона, 177

выбор системы отсчета, 272

высота

пирамида проекта, 140

вычисления

культура отношения к результатам, 52

проверка, 53

точность, 49

G

Галилей, 178

гейзенбаг, 276

Гейзенберг, принцип

неопределенности, 276

генератор случайных чисел

игральные кости как, 296

Гешпенгеймер, Т.А.

см. Herppenheim, T. A.

Гизан, Оливье *см.* Guisan, Olivier

гонка в космосе 1960-х г., 41

график работ, 167

последняя неделя, 174

график степени завершенности проекта, 182
график суммарной силы, действующей на проект, 195
график человеческого поведения, 182
графическое представление информации, 102

Д

дифференциальное исчисление, 178
длинные векторы, 87
длительность итерации в неделях как квадратный корень из длины кода, 164
достоверность и точность оценок, 159

Ж

железный треугольник «ресурсы, охват и время», 137
жертва большого проекта, 236
жизненный цикл программного продукта, 283

З

завышение оценок времени, 158
закон Ома, 108
закон Паркинсона, 93
законы движения Ньютона, 268

И

игра в «два графика», 167
игра в животных (стандартная задача), 115
игра в «подковки», 84
игральные кости
 как генератор случайных чисел, 298
имплантируемый кардиодефибриллятор, 31
инженерная дисциплина в программировании, 33
инстинкты в управлении командами, 76
информация, графическое представление, 102
использование аналогий, 266
итеративная разработка
 и прототипы, 91
 и ритм проекта, 188
 и сборка, 131

 как противоядие для отставания от графика, 168
 обоснование, 99
 обучение на практике, 94
итеративный подход
 начало и завершение, 39
 циферблат для решения задач, 36
 этапы, 36, 38
итерации, 87
 оценка длины, 166

К

калькулятор для инженеров HP-35, 47
каскадная и итеративная разработка
 следствия для бизнеса, 95
 эффект численности персонала, 96
качество
 пирамида проекта, 139
 программного обеспечения, 32
качество продукта, оценка, 151
квадратичная сложность алгоритмов, 232
квантовая механика, 274
Кеннеди, Джон *см.* Kennedy, John
Керниган и Ритчи *см.* Kernighan, Brian W. & Ritchie, Dennis M.
«клог» (kludge), 262
код, подсчет количества строк, 164
код для проверки правильности входных данных, 51
команды
 определение, 69
 подбор участников, 57
 управление, 69
компромиссы программного проекта, 135
конкурентная борьба как причина неудачных решений, 64
константа тонкой структуры, 275
контроль и учет, 60
контрольные точки, 60
конус адекватного вознаграждения, 247
 девять возможных случаев, 250
 достижение обоюдного удовлетворения, 253
 нестабильные диагональные состояния, 252
 распределение кандидатов в команду по классам, 255

распределение участников команды по классам, 254
 короткие векторы, 86
 разработка программного обеспечения, 89
 корпоративные ценности, 344
 кривая выполнения проекта, 177
 кривая обучения, 183
 кривая скорости проекта, 184
 кривые обучения и завершения, 192
 критически важные программы, 32
 Круктен, Филипп *см.* Kruchten, Philippe
 культура
 влияние крупных клиентов на, 353
 и ценности в бизнесе, 341
 и поиск работы, 355
 определение, 341
 преемственность, 350
 сильный и слабый типы, 343
 создание, обязанности руководства, 350
 угрозы преемственности, 351
 культурный контекст в политике, 200

Л

Лав, Том *см.* Love, Tom
 Ларсен Говард *см.* Larsen, Howard
 лидерство и управление, 70
 лидеры, 70
 логарифмическая линейка, 47, 51
 логнормальное распределение, 144

М

Маккиавелли, 208
 Мараско, Рослин Розенталь *см.* Marasco, Roslyn Rosenthal
 Мараско, Эндрю *см.* Marasco, Andrew
 масштабируемость, 91
 «махер», 357, 361
 машина Тьюринга, 281
 менеджеры
 изучение новых языков программирования, 112
 изучение новых языков программирования путем написания программ, 113
 менеджеры проектов
 калибровка умения оценивать, 173
 менеджеры: «страусы» и «ленивцы», 71

«менш», 357, 363
 Мерецки, Уэйн *см.* Meretsky, Wayne
 метрики обучения и завершения в методологии Rational Unified Process, 191
 «Мифический человеко-месяц», книга Ф. Брукса, 150
 «множество правил и никакой милости», жесткий процесс, 127
 моделирование
 UML (Unified Modeling Language), 103
 модель проекта
 пирамида, 139
 мультипликативная центральная предельная теорема, 145
 Мур, Джефффри *см.* Moore, Geoffrey A.

Н

наука и техника
 представление в общественном сознании, 43
 несоответствие импедансов, 221
 новые направления деятельности
 влияние на культуру, 354
 номограммы, 337
 нормальный закон распределения, 142

О

область действия теории Эйнштейна и законов Ньютона, 272
 область применения принципа Гейзенберга, 278
 обновление или замена программного обеспечения, 283
 для мобильных устройств, 285
 стимулы для, 287
 оборонительный стиль программирования, 51
 обследование, фаза проекта, 190
 обучение на практике, 90
 обучение студентов инженерных специальностей в 1960-1970 г., 45
 общая теория относительности
 экспериментальные проверки, 274
 общение с инженерами, 215
 не предлагать решение задачи, 218
 постановка задачи, 217
 права собственности на задачи, 217
 установка срока решения задачи, 220

- общие принципы управления, 35
- обязательства
 - нарушение из-за неверной оценки трудоемкости задач, 237
 - неоднозначно понимаемые, 229
 - реалистичные, 239
 - три отговорки при невыполнении, 232
- опыт как критерий отбора участников команды, 57
- организация команды, 58
- ориентация на клиента, 346
- отгрузка новой версии продукта, 121
- охват проекта, пирамида, 139
- «охват, ресурсы, время – выбери два пункта», 136
- оценка
 - достоверность и точность, 159
 - задержки проекта, 170
 - результата вычислений предварительная, 48
- оценка времени, 157
- оценка времени проекта, динамическая, 170
- ошибка на порядок при расчетах на логарифмической линейке, 48
- ошибочное применение законов Ньютона
 - второго, 270
 - первого, 269
 - третьего, 271
- П**
- пакетное выполнение программ, 50
- передача в эксплуатацию, фаза проекта, 190
- Перроу, Майк *см.* Perrow, Mike, 202
- песочница, 123
- пирамида проекта
 - объем, 140
 - ограничения модели, 149
 - взаимозависимость параметров, 150
 - неитеративный подход, 152
 - неравноценность параметров, 150
 - условность постоянства объема, 154
- управление вероятностью успеха проекта, 147
- учет риска, 154
- планирование
 - контроль и учет, 60
 - контрольные точки, 60
 - общая цель, 62
 - организация команды, 58
 - решительность, 62
 - риски, 61
 - составление календарных планов, 59
 - уяснение объема задач, 56
- плохая политика, разновидности, 210
- повторяемый процесс сборки, 123
- подбор команды разработчиков, 57
- поиск работы
 - учет культуры и ценностей компании, 355
- ползучий фичеризм, 136
- политика, 199
 - и компромиссы, 204
 - как часть окружающего мира, 126
 - культурный контекст, 200
 - нежелание подчиниться, 210
 - нейтральная, 206
 - плохая, 207
 - строительство империи, 211
 - хорошая, 205
- политический процесс
 - определение, 201
 - три сценария, 202
- получения одобрения от всех участников проекта, 226
- последовательный подход (водопад-ный процесс)), 81
- постоянная Планка, 278
- постоянство
 - объема пирамиды проекта, 140
 - построение системы, фаза проекта, 190
 - потока состояние, 241
 - предсказуемость графиков работ, 174
 - презентации PowerPoint, 102, 226
 - принцип неопределенности Гейзенберга, 276
- принципы управления
 - общие и специальные, 35
- принятие нового продукта рынком
 - S-кривая, 184
- приоритеты рисков, 93

причины провала программных проектов, 65
 причины успеха программных проектов, 66
 провал проекта
 в результате накопления всех невыполненных обязательств, 235
 программирование рисков, 91
 программное обеспечение
 значение качества, 32
 обновление или замена, 283
 программные проекты, человеческий фактор, 67
 программы, критически важные, 32
 продуктивность, оптимальная в потоке, 242
 проекты
 важность тестирования, 264
 добавление новых людей в опаздывающие, 320
 компромисс между стоимостью и ускорением работы, 329
 кривая выполнения, 177
 преимущество руководства, 263
 принятие на себя руководства чужим, 313
 сосредоточенность на второстепенных деталях, 263
 проработка, фаза проекта, 190
 просчеты составления графиков
 причины, 167
 прототип и продукт, 264
 прототипы, 91

Р

разбор кода, 101
 разработка программного обеспечения
 короткие векторы, 89
 расползание проекта, 178
 распределение вероятности исхода
 программного проекта, 142
 логнормальное распределение, 144
 расширение организации, 319
 влияние на культуру, 351
 возникающие проблемы, 320
 модель, 323
 нелинейная зависимость новой продуктивности от параметров, 332

номограмма для уравнений модели, 337
 польза от новых работников, 322
 продуктивность в переходный период, 321
 рекомендации для подбора сотрудников, 334
 электронная таблица для расчета модели, 339
 религиозные войны, 128
 риски, 61, 138, 154
 приоритеты, 93
 ритм проекта, 176
 S-кривая, 183
 и итеративная разработка, 188
 Ройс, Уокер *см.* Royce, Walker
 роль политики, 199

С

сборка продукта
 инструментарий, 129
 итеративная разработка и, 131
 политика, 125
 процесс, 127
 трудности, 124
 синдром «паралича при анализе», 62
 система образования, 42
 в США, реформы 60-х г., 43
 скорость, 178
 пирамида проекта, 139
 слияния и приобретения
 влияние на культуру, 352
 сложность задачи
 несоответствие уровню мастерства, 243
 сосредоточенность на клиенте, 70, 349
 составление календарных планов, 59
 спутник, запуск Советским Союзом, 41
 и система образования, 42
 среда с высокой степенью доверия, 203, 209
 стандартная задача для изучения языка программирования, 113
 стандартная задача
 игра в животных, 115
 стандартное нормальное распределение, 143
 строки
 подсчет количества в коде, 164

Сэдлер, Марк *см.* Sadler, Mark

Т

талант делать случайные открытия, 303
теорема Геделя о неполноте, 281
теория относительности Эйнштейна, 272
тепловая смерть, 278
термодинамика, 279
технические специалисты
и политика, 200
этические нормы, 208
траектория полета шарика, подброшен-
ного вертикально, 179

У

Уайдман, Макс *см.* Wideman, Max
универсальная гравитационная
постоянная, 276
универсальная система обозначений,
103
унифицированный язык моделирова-
ния *см.* UML
унифицированное управление измене-
ниями *см.* UCM
Уокер, Джон *см.* Walker, John
упаковка, при сборке продукта, 125
управление командами, 69
внимание к мнению участников, 72
готовность встретить проблемы, 71
доверие инстинктам, 76
злонамеренность или некомпетент-
ность, 74
лидерство и управление, 70
сдержанность, 73
сосредоточенность на фактах, 72
юмор, 75
управление охватом проекта, 136
Урми, Яаак, *см.* Urmi, Jaak
ускорение, 178
шарика, подброшенного вертикаль-
но, 181
успех проекта
определение, 149
примат времени, 136
причины, 66
психология, 150
учет рисков, 61

уяснение объема задач при
планировании, 56

Ф

фазы проекта в методологии Rational
Unified Process, 190
фазы развития человека, 357
Фейнман, Ричард *см.* Feynman, Richard

Х

харизма, 70
Хемминг, Ричард *см.* Hamming, Richard

Ц

ценности
корпоративные, 344
честность, сосредоточенность на ин-
тересах клиента и результатах, 345
центральная предельная теорема, 153
циферблат для итеративного решения
задач, 36

Ч

человеческий фактор, 67
честность, 348
четыре шага общения с инженерами-
программистами, 216
Чиксентмихайи, Михай *см.* Csikszentmi-
halyi, Mihaly
численность персонала при каскадной
и итеративной разработке, 96

Ш

«шлеппер», 357, 358

Э

экономия
пирамида проекта, 139
эксперименты
двойной слепой метод, 277
энтропия, 124, 279
в замкнутой системе, 279
этапы
второго цикла итеративного
подхода, 38
первого цикла итеративного подхо-
да, 36

этические нормы
у технических специалистов, 208
эффект обособления, 158
эффективность труда и вознагражде-
ние, 240

Я

языки программирования
изучение новых, 112
изучение путем написания про-
грамм, 113
ограниченное применение на прак-
тике, 120

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-096-0, название «IT-проекты: фронтовые очерки» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.