БИБЛИОТЕКА ПРОГРАММИСТА

Дж. Ханк Рейнвотер

Как пасти котов

Наставление для программистов, руководящих другими программистами

- Как вести за собой тех, кто привык гулять сам по себе
- Как стать лидером
- Как выявлять и устранять возможные проблемы
- Как общаться с топ-менеджерами

apress®

ПИТЕР®

J. Hank Rainwater

Herding Cats:

A Primer for Programmers Who Lead Programmers

Apress®

БИБЛИОТЕКА ПРОГРАММИСТА

Дж. Ханк Рейнвотер

Как пасти котов

Наставление для программистов, руководящих другими программистами

Москва · Санкт-Петербург · Нижний Новгород · Воронеж
Киев · Екатеринбург · Самара · Минск

2016

ББК 65.290-23
УДК 658.014.1
Р35

Дж. Рейнвотер

Р35 Как пасти котов. Наставление для программистов, руководящих другими программистами. — СПб.: Питер, 2016. — 256 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-496-01820-3

«Как пасти котов» — это книга о лидерстве и руководстве, о том, как первое совмещать со вторым. Это, если хотите, словарь трудных случаев управления IT-проектами. Программист подобен кошке, которая гуляет сама по себе. Так уж исторически сложилось. Именно поэтому так непросто быть руководителем команды программистов. Даже если вы еще месяц назад были блестящим и дисциплинированным программистом и вдруг оказались в роли менеджера, вряд ли вы знаете, с чего надо начать, какой выбрать стиль руководства, как нанимать и увольнять сотрудников, проводить совещания, добиваться своевременного выполнения задач. В таком случае без этой книги вам не обойтись. А может быть, вы — опытный менеджер, желающий пересмотреть свои принципы лидерства? Тогда, опять же, эта книга для вас. Вне зависимости от возраста, пола и социального статуса она поможет вам укрепить свои позиции в роли лидера программистов. Материал изложен довольно компактно и легко укладывается в голове. Стоя в книжном магазине и раздумывая, что же купить, задайте себе один простой вопрос: «Нужно ли мне совершенствовать свои лидерские навыки?» Полагаю, вы ответите: «Да», — а значит, данная книга окажется для вас небесполезной.

12+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 65.290-23
УДК 658.014.1

ISBN 1590590171 англ.
978-5-496-01820-3

Краткое содержание

Содержание

Посвящается Дэвиду, моему любимому сыну, —
память о тебе меня неизменно вдохновляет.
Жаль, что тебя нет,
и я никогда больше не увижу, как ты смеешься...

Предисловие

Прочитав замечательную книгу Хэнка, в которой он рассказывает о выпасе котов, я вспомнил то время (а было это... ну очень давно), когда из программиста меня перевели в менеджеры. Подобно вам, читатели, я был в высшей степени самоуверенным программистом-аналитиком. Я специализировался на языке PLI и базах данных IMS DB/DC. Прибавить к этому понемногу Ramis, FOCUS, Easytrieve Plus, Datacom/IDEAL, CICS, VSAM — и получится вполне сформировавшийся программист, пишущий для мэйнфреймов. Сегодняшние кодировщики могут с полным правом относить эти технологии к древней истории, но, смею вас уверить, в те времена по крайней мере некоторые из них были очень даже ничего!

Подобно Хэнку, я сначала взял на себя неофициальные обязанности координатора и наставника своих коллег — в основном, молодых сотрудников. Впоследствии эти полномочия были закреплены за мной уже формально. Таким образом, я начал сочетать полную ставку программиста-аналитика с полноценным руководством. После этого мне открылись как положительные, так и отрицательные стороны менеджмента. Помню, собрались мы — я и еще несколько таких же менеджеров — как-то раз на совещание с начальником. Начальник говорил о том, какие у него на наш счет большие ожидания, о необходимости повышать нашу квалификацию как руководителей. Одна из моих коллег в ответной речи выразила крайне распространенную среди молодых менеджеров позицию. Она заявила, что могла бы стать значительно лучше как руководитель, будь у нее в подчинении более приемлемый персонал.

Не всегда понятно, почему некоторые вещи крепко оседают в памяти на долгие годы, и как раз ее фразу я никак не могу забыть. Полагаю, будь в нашем распоряжении тогда учебник вроде того, что написал Хэнк, переход к менеджерским обязанностям прошел бы значительно менее болезненно. В конце концов, кто мы такие были? Программисты, которым поручили координировать деятельность других программистов. В результате былые приятельские отношения испарились — будто их никогда и не было. Мне совершенно не хотелось менять свое отношение к коллегам, но без этого контролировать их поведение я не мог. Мы остались друзьями, но эти отношения перешли на другой уровень, что ли, и назад пути уже не было.

Сегодня, по прошествии многих лет, я счастлив, что в роли лидера команды разработчиков, руководителя проектов, руководителя группы, руководителя отдела и директора мне приходится координировать действия более чем 200 людей. За счет посещения разного рода курсов и семинаров мне удалось усовершенство-

вать навыки руководителя. Наконец, слава богу, что пользу от чтения книг по менеджменту я осознал довольно рано. В конце концов, личность в сегодняшних условиях — это опыт плюс прочитанная литература.

Мой опыт перехода из программистов в руководители позволяет в полной мере оценить предложения, высказанные Хэнком в его книге. Его стараниями любой специалист, находящийся в аналогичном положении, может рассчитывать на существенную помощь. Уже в первой главе Хэнк попадает в десятку утверждением: «то, что делаешь ты, не обязательно буду делать я». В самом деле, не с этим ли связаны все те разочарования, которые мы испытываем в период адаптации к роли руководителя? Если вы принимаете эту проблему близко к сердцу, поверьте мне — Хэнк поможет вам преодолеть подобного рода затруднения.

Попробую перефразировать мою бывшую коллегу: заниматься менеджментом было бы значительно проще, если бы все подчиненные были как две капли воды похожи на своего начальника. К счастью, это не так. Люди руководствуются разными мотивами, у них разный уровень знаний, и понять, что движет тем или иным деятелем, не так-то просто. Различия не превозносят одного человека над другим — просто все мы разные. Что делает руководитель? Он координирует и ведет всех этих «котов», которые гуляют сами по себе. Понимать, как коты себя ведут и как общаются между собой, совершенно необходимо — иначе эффективного лидерства не получится.

Вспоминается мой разговор с начальником о трудностях, причиной которых стал еще один руководитель из числа бывших программистов. Начальник тогда заявил мне буквально следующее: «Том, пока я сижу на этом месте, никто из программистов больше не станет менеджером!» Где-то через год, во время моего отчета перед новым начальником, мы принялись обсуждать одного из технических руководителей, которому удалось добиться поразительных успехов по части организации и мотивирования своих сотрудников. Этот начальник резюмировал свои соображения так: «Том, я думаю, что впредь всех руководителей нам следует набирать из числа технарей».

Эти диаметрально противоположные точки зрения лишний раз доказывают, что нет двух совершенно одинаковых людей. У всех разные таланты, способности, желания и наклонности. Вы, помимо других, должны разобраться в собственных достоинствах и недостатках (см. главу 2) и задействовать свои навыки таким образом, чтобы обеспечить успешную деятельность группы в целом (см. главу 3). Программисты, которым приходится впервые брать на себя обязанности по руководству другими программистами, обнаруживают себя на перекрестье профессий. Некоторые на постоянной основе переходят к менеджерской деятельности. Другие склоняются к программированию, поскольку в этой области от них больше толку. Остальных устраивает промежуточная позиция между руководителем и кодировщиком.

Если оставить в этой книге только три первые главы, а все остальное выкинуть, все равно ее стоило бы прочесть (ее объем, конечно, сильно бы уменьшился). Однако и остальные главы тоже весьма интересны, поскольку Хэнк рассматривает в них чрезвычайно актуальные для молодых руководителей вопросы.

С одной стороны, он говорит о том, что теперь у вас есть формальные административные обязанности. Навыки руководства людьми в контексте успешной деятельности компании очень важны, но, с другой стороны, именно административные

вопросы обеспечивают плавное вращение коммерческого маховика. Вам предстоит постоянно заниматься поиском информации и составлять рецензии на выполненные задания. В рамках подведомственной группы вы находитесь на вершине иерархии управления. Лишь за счет дисциплины и самоорганизации люди справляются с административными функциями. Если эти качества в вашем характере отсутствуют, вы станете слабым звеном административного механизма, и ваш собственный начальник будет вынужден постоянно вас подталкивать. Скажите спасибо Хэнку за объяснение стандартной роли администратора — это объяснение помогает понять, что такое же бремя несут многие ваши коллеги.

Глава 5, посвященная проведению совещаний, затрагивает очевидно недооцененный комплекс приемов. Сама постановка вопроса о продуманной организации совещаний заслуживает уважения. Случалось ли вам посещать совещания, не преследующие конкретной цели и никем не управляемые? (Вероятно, этот вопрос лучше переформулировать так: «Каков процент посещенных вами совещаний, которые проводились подобным образом?») Если случалось, теперь вы знаете, почему так происходит: на этих совещаниях никто не проявил активной руководящей позиции. Если вам удастся при проведении совещаний взять на себя лидерство, сделав их тем самым более продуктивными и сориентированными на конкретные задачи, скажите спасибо Хэнку.

Еще один раздел книги, который мне очень понравился, посвящен отношениям с начальством (см. главу 9). Ориентироваться в данной области очень важно, хотя многие слишком поздно осознают это обстоятельство. Да, действительно, ваш начальник должен руководить вами. В то же время активная роль в выстраивании ваших отношений может принадлежать вам. Занятно, но если вы нацелитесь на то, чтобы успех был достигнут вашими подчиненными и начальством, успех обязательно придет к вам самому.

У меня нет возможности углубляться в содержание всех глав и излагать свои соображения по поводу собранного в них материала — в противном случае мое предисловие грозит превысить по объему саму книгу. (Впрочем, позволю себе одно, последнее замечание: обязательно прочтите разделы о многонациональных и распределенных группах. Крайне ценные сведения!) Скажу лишь, что эта книга может существенно облегчить жизнь тех программистов, которые в один прекрасный день обнаруживают себя на руководящих постах. Аналогия с выпасом котов, по-моему, вполне уместна. У многих видов животных развито стадное чувство, и пасти их, соответственно, не так уж трудно. У меня дома два кота, и я на своем опыте знаю, что у них этот инстинкт не просматривается. Вести в заданном направлении даже одного кота (программиста) — нетривиальная задача. А для того чтобы вести за собой четыре-пять (или дюжину) этих упрямых тварей, требуется предельная концентрация и весьма специфический комплекс приемов. На материале этой книги, я надеюсь, вы сможете ознакомиться с требованиями, которые обычно предъявляют к людям, исполняющим вашу роль, и значительно упростить для себя путь к успеху — по крайней мере, опыт Хэнка к этому располагает!

Том Мокел (Tom Mochal),
создатель www.TenStep.com,
президент TenStep, Inc.

Об авторе

Хэнк Рейнуотер (Hank Rainwater) в настоящее время работает в Risk Sciences Group (Атланта, Джорджия), где руководит группой программистов, разрабатывающих программные продукты для страховых компаний. Его путь в науке и инженерии насчитывает более трех десятилетий. В разные периоды жизни он занимался программированием на языке Фортран с использованием перфокарт; преподаванием математики в колледже; исследованиями в областях радиоастрономии, систем наведения ракет и телеметрических систем; координацией производства встроенных систем цифрового управления. Как специалист в сфере разработки программных продуктов Хэнк успел поработать консультантом, лектором, программистом и руководителем групп разработки программ для самых разных областей человеческой деятельности. Что касается образования, Хэнк окончил колледж с физическим уклоном и получил диплом университета по специальности «математика и физика». Кроме того, Хэнк — магистр теологии. Он несколько лет был пастором в разных приходах (как в США, так и за рубежом), занимаясь попутно преподаванием теологических дисциплин.

Хэнк убежден, что основным условием успешного руководства программистами является наличие лидерских навыков. До прихода в индустрию разработки программных средств он не понимал истинного значения этого утверждения. С опытом он стал более раскрепощенным, отрастил длинные волосы

и теперь получает нескрываемое удовлетворение от общения с творческими личностями, старающимися превратить код в продукты с радужными рыночными перспективами.

Без шевелюры его трудно выделить из уличной толпы.

О научном редакторе

Дэйв Кристенсен (Dave Christensen) в настоящее время трудится на посту старшего системно-технического аналитика в целлюлозно-бумажном отделении корпорации Potlatch (штаб-квартира находится в Клокете, штат Миннесота). В его обязанности входит обеспечение компании конкурентных преимуществ за счет разрабатываемых им веб-приложений. Кроме того, он — президент Proxis Productions (http://www.proxis-productions.com) — консалтинговой компании, специализирующейся на проектировании распределенных корпоративных веб-приложений. В 1995 году, когда компания Proxis Productions только создавалась, Дэйв предполагал заняться компьютерной графикой для видеоигр и других коммерческих предприятий, однако с утверждением графики в Интернете он получил возможность свести свои графические и технические интересы воедино. У Дэйва диплом по английской литературе; кроме того, он прослушал несколько медицинских курсов и занимался в театре в колледже St. Scholastica. Дэйв — счастливый человек: свободное от работы время он проводит с прекрасной женой и двумя изумительными детьми, которые не перестают его удивлять. У них много животных: две собаки и выводок кошек, которые, кстати, тоже поучаствовали в работе над этой книгой. Кроме того, Дэйв — коллекционер редких рыбок и увлеченный реставратор. Он обожает раскрывать в окружающих скрытый потенциал. В этом отношении он успел поэкспериментировать на автомобилях, домах и людях.

Об иллюстраторе

Мелани Уэллс (Melanie Wells) — опытный графический дизайнер с десятилетним опытом работы в самых разных областях: иллюстрировании книг, разработке корпоративных логотипов, создании брошюр, проектировании выставочных стендов, оформлении почтовой атрибутики, журнальной рекламы, каталогов, дизайне упаковки, веб-сайтов, спортивной одежды, товарном дизайне и т. д.

Помимо собственно графического дизайна, Мелани увлекается изобразительным искусством. Вне зависимости от текущего занятия — разработки фирменного стиля или, например, рисования маслом на холсте — она в первую очередь художник.

Благодарности

Огромное спасибо Дэну Эплмену (Dan Appleman) из Apress — прочитав черновик первой главы, он по достоинству оценил замысел книги, в которой, в отличие от многих, не рассматривается ни один из современных языков программирования. Ты молодец, Дэн! Уже много лет ты вдохновляешь меня (и не только меня) на новые достижения, и я надеюсь в этом отношении пойти по твоим стопам.

Карен Уоттерсон (Karen Watterson) — человек, чьи рекомендации в период работы над проектом сыграли решающую роль в формировании многих идей, которые ранее находились в зародышевом состоянии. Твои обширнейшие знания, опыт и письма, приходившие в любое время дня и ночи, очень помогли мне. Спасибо, Карен!

Это мой первый опыт на поприще написания книг. В связи с этим не могу не упомянуть Трейси Браун Коллинз (Tracy Brown Collins) — как руководитель проекта она проявила себя с лучшей стороны. Спасибо тебе за проницательные литературные замечания и за то, что благодаря тебе вся наша команда успешно сработалась.

Дэйв Кристенсен — мой главный научный редактор — помог мне разобраться со стилем изложения. Несколько раз я все же оплошал, но с его помощью многие огрехи удалось ликвидировать. Кроме того, я обширно «одалживал» его комментарии. Спасибо тебе, Дэйв, за недюжинные старания.

Я очень признателен Николь Леклерк (Nicole LeClerc) из Apress за грамматическую правку. Мы, программисты, далеко не всегда оказываемся на высоте, сталкиваясь со сложными синтаксическими конструкциями. Куда лучше мы ориентируемся в хитром и загадочном стиле кодирования. Николь восполнила мои школьные пробелы в знаниях касательно орфографии, пунктуации и других мелочей, за счет которых эту книгу стало значительно проще читать.

Джессика Лендисман (Jessica Landisman) провела долгие часы за аннотированием, корректурой, составлением предложений и комментированием первых черновиков. Как опытному высококвалифицированному программисту ей нет равных, и я действительно чрезвычайно благодарен за оказанную мне помощь.

Отдельное «мерси» Кэти Хейнс (Kathy Haynes), корпевшей над рукописью на разных стадиях ее готовности, причем каждый раз после ее участия текст становился неизмеримо лучше. Для меня она, уроженка Восточной Алабамы, — непререкаемый авторитет по диалекту американского Юга.

Все восторги по поводу графического оформления этой книги прошу перенаправлять Мелани Уэллс — замечательной художнице, работающей с самыми разными материалами. У нее настоящий талант на выражение мыслей в визуальных образах. Благодаря тебе, Мелани, коты вышли — круче некуда!

Наконец, я глубоко признателен своему отцу Джулиусу Рейнуотеру, набросавшему кое-какие из иллюстраций, ныне украшающие страницы этой книги. Инженер, предприниматель, замечательный отец, несомненный образец для подражания, обладатель многочисленных талантов — я перед тобой в неоплатном долгу. Ну а без тебя, мама, я вообще ничего не смог бы сделать — даже написать эту книгу!

От издательства

Ваши замечания, предложения и вопросы отправляйте по адресу электронной почты comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

Все исходные тексты, приведенные в книге, вы можете найти по адресу http://www.piter.com/download.

Подробную информацию о наших книгах вы найдете на веб-сайте издательства: http://www.piter.com.

Введение

Правды нет — о ней нам только рассказывают.

Анонимный автор с Юга

В этой книге я намерен рассмотреть весь комплекс задач, стоящих перед руководителем. Мужайтесь: решить эти задачи вполне реально, что бы вам ни говорили. От вас требуется выстроить стройную систему мышления, эдакий гештальт. Что такое «гештальт»? Согласно словарю Вебстера, это «структура со свойствами, которые невозможно получить путем простого сложения ее элементов». Если перевести это определение на язык объектно-ориентированного программирования, которым, полагаю, вы владеете лучше, чем стилем изложения Вебстера, получится следующее: вашей мысленной архитектуре предстоит пройти серьезные преобразования. Унаследовав навыки менеджера, вы должны будете перегрузить типичные параметры мышления новыми типами и значениями — то есть, грубо говоря, испытать полиморфизм собственного характера. За счет этого вы инкапсулируете в своем программистском мозгу совершенно новый вид искусства — искусство лидерства и руководства. Между руководством и лидерством прослеживаются существенные различия. Нужно то и другое, но лидерство все же имеет приоритет — хотя крепкие руководящие навыки, естественно, помогают брать новые высоты в роли лидера.

Своевременность производства и качество программных продуктов вашей компании — теперь в ваших руках, и смею надеяться, что моя книга внесет некоторое разнообразие в вашу деятельность. Нельзя же, в конце концов, проводить все рабочие дни совершенно одинаково!

Структура книги

Посмотрим, какие сведения можно почерпнуть из составивших книгу глав.

Глава 1. Как привыкнуть к роли руководителя.

Для развития лидерских качеств нужны новые приемы — навыков, наработанных в бытность программистом, вам не хватит. В этой главе мы поговорим о том, как адаптироваться к новой должности. Для этого я составил перечень наиболее распространенных личностных типов программистов, которые, естественно, оказывают то или иное влияние на способность управлять процессом раз-

работки и направлять его по заданному курсу. Вам надлежит осознать поразительную вариативность характеров подчиненных, попытаться проанализировать их личные качества и найти к ним индивидуальные подходы. В конце концов, вы же главный — что ж тут плохого?

Глава 2. Как руководить собой.

Здесь вам придется добраться до глубин своего сознания (не бойтесь — это не так страшно) и самолично усвоить принципы руководства. *Если вы не научитесь управлять собой, занять лидерскую позицию среди коллег не получится.* Как говорил Уинстон Черчилль: «Чем пристальнее мы вглядываемся в прошлое, тем проницательнее становимся, предсказывая будущее». Этот афоризм рекомендую применить к вопросам самоанализа.

Глава 3. Как вести стаю за собой.

Лидерская роль предполагает приобретение новых навыков вдобавок к навыкам чисто программистским. В этой главе дается обзор основных областей деятельности лидера, на которые следует обратить особое внимание. В противном случае вы рискуете, поддавшись внешним влияниям, пойти в неверном направлении, а сотрудники группы, подобно испуганным котам, от вас разбегутся. Мне совершенно не хочется, чтобы вы, как говаривал лорд Байрон, оказались среди тех «немногих, чьи души всплывают после крушения надежд».

Глава 4. Как организовать успех.

Здесь мы прервем на некоторое время дискуссию о взаимоотношениях с окружающими. Повысив уровень личностной организации, вы сможете взять новые высоты по административной части. Кроме того, я советую вам изучить организационную структуру своей компании и изыскать способы повышения эффективности работы. Таким способом вы сможете выделить время на развитие лидерских качеств — иначе говоря, на выполнение своих основных обязанностей.

Глава 5. Как вести совещания.

В бытность программистом вы, вероятно, привыкли не советоваться ни с кем, кроме самого себя. Теперь эту ситуацию придется менять. Никаких более совещаний во время утреннего бритья и любования на красавца в зеркале! Вам предстоит обсуждать дальнейшие действия с себе подобными (разве что не такими симпатичными, как вы) и, что гораздо страшнее, с людьми, которые, как это ни странно, *не зарабатывают на жизнь кодированием*! В роли лидера на совещаниях от вас потребуется терпение. Не отчаивайтесь и не забывайте слова Леонардо да Винчи: «Нетерпеливость — мать глупости».

Глава 6. Философия и методы технического лидера.

В этой главе я рассмотрю некоторые технические принципы и их философские обоснования. Одно дело принимать технические решения применительно к собственному кодовому заданию, и совсем другое — делать это за весь отдел. Вполне возможно, что вы успели взойти на экспертный уровень в области технологии, но это не отменяет необходимости анализа последствий принятия технических решений в корпоративном масштабе. Здесь мы обговорим вопросы архитектуры, проектирования и критических обзоров кода.

Глава 7. Закат лидера.

Все руководители (не только вы, но и ваше начальство) подвержены влиянию упадочных стратегий лидерства, и иногда мы, к сожалению, этому влиянию действительно поддаемся. Некоторые стили руководства не допускают конструктивного лидерства, а значит, их следует избегать. Здесь я опишу возможные варианты деградации лидерских качеств вследствие принятия неверной стратегии и попутно предложу способы выхода из кризиса.

Глава 8. Восход лидера.

Подобно программным продуктам, которые конструируются на основе надежной архитектуры, лидерские качества культивируются на основе присущих лидеру черт характера. В этой главе я попытаюсь свести все аспекты лидерства воедино. Если перефразировать Эмерсона, «многословие — бедствие для авторов, поощряемое издателями, читателями и книготорговцами». Что важнее, здесь я излагаю базовые принципы успешного лидерства и демонстрирую методы их настройки как необходимое условие профессиональности руководства.

Глава 9. Как ужиться с начальством.

Обратите внимание: глава называется «Как ужиться с начальством», а не «Как руководить начальством», — последнее просто не представляется возможным. Тем не менее налаживать отношения с теми сотрудниками, которым вы подотчетны, следует не менее тщательно, чем с собственными подчиненными. Субординация — это совсем не пустое слово. Здесь мы детально обговорим методы формирования слаженной команды из двух человек: вас и вашего босса.

Глава 10. Слова без песни.

В этой главе раскрываются самые разные, порой не связанные друг с другом, темы, которые не всегда касаются ежедневных обязанностей лидера программистов, но, тем не менее, представляют в контексте выпаса котов немалую важность. Руководство распределенной группой разработчиков, оценка тенденций развития методологий разработки программных средств и некоторые другие темы рассмотрены в этой главе. С ее помощью, надеюсь, вам будет проще превратить хаос в порядок и не сойти при этом с ума.

Послесловие. Снова в плавание…

В заключение я извергну из глубин сознания несколько мудрых наставлений — по крайней мере, в нашем сдвинутом, но оттого не менее прекрасном мире разработки программного обеспечения мои слова, полагаю, сойдут за непреходящую мудрость.

Приложение А. Как ухаживать за живностью — электронный администратор.

В этом приложении речь пойдет о программе электронного администратора, которая упоминается в главе 4. Мне совершенно не жалко делиться с вами кодом — надеюсь, что мои идеи не совсем бесполезны, и с их помощью вы сможете

доработать решение под себя. Любой обладатель этой книги, написав секретное слово, сможет скопировать код с сайта книги. Впоследствии, по мере материализации моих идей в коде, у вас появится возможность оценить степень их полезности.

Приложение Б. Как дать скотине в глаз — критический обзор кода электронного администратора.

Основной предмет обсуждения здесь — способы практического применения принципов критического обзора кода, рассматриваемых в главе 6, к программе, описанной в главе 4 и приложении А. Хотя мне приходится оценивать собственный код, я стараюсь соблюдать объективность, указывая на его достоинства и недостатки. Тем самым я намерен продемонстрировать вам, как в реальных условиях осуществляются функции кодового полицейского.

Библиография. Ресурсы для специалистов по выпасу котов.

В библиографии фигурируют все работы, на которые по мере изложения материала я счел нужным сослаться, а также несколько других, заслуживающих внимания с вашей стороны. Некоторым участникам нашей лидерской гонки удалось вырваться далеко вперед, и я намерен вас с ними познакомить.

Алфавитный указатель.

Вы знаете, для чего нужна эта штука. Сюда можно заглянуть в поисках отдельных понятий или имен, чтобы не читать книгу целиком. Примерно по тому же принципу обучаются языкам программирования по справочникам. Они дают представление о синтаксисе, но в отсутствие контекста весьма высока вероятность концептуальных ошибок.

По всему тексту разбросаны небольшие врезки под общим заголовком «Кошачьи разборки». В них я рассказываю реальные истории из жизни руководителей программистов, призванные поучать вас и освещать явно неудачную практику. Эти врезки иллюстрируют многие из представленных в книге принципов. Имена, места, языки программирования и типы приложений, конечно, изменены — с тем, чтобы защитить невинных и скрыть виновных. Эти истории легко читаются, а самое главное, наглядно демонстрируют примеры неверных действий. Похоже, мы действительно лучше всего учимся на собственных ошибках. Здесь же, рассказывая об ошибках, совершенных другими людьми, я пытаюсь уберечь вас от повторения их печального опыта. Вспомним Альбера Камю: «Глупы те, кто считает, что пессимистические мысли порождают отчаяние». За счет законов физики мы способны превратить отрицательный опыт в обнадеживающие перспективы — иными словами, опыт окружающих должен подтолкнуть вас к движению в правильном направлении.

Есть и другие врезки, в которых выделены основные положения текущего текста. Такой вариант верстки поможет вам изрядно сэкономить — теперь вам не нужно разоряться на желтый маркер и выделять ключевые моменты изложения. Хотя нет: прибегайте к маркеру ровно столько раз, сколько потребуется. По-моему, самое важное в том, что эта книга открывает простор для осмысления ваших рабочих функций. Чем больше полезного вы в ней найдете, тем лучше — и мне и вам.

Кому и зачем стоит прочесть эту книгу

От чтения этой книги выиграют программисты, превратившиеся в менеджеров, руководителей групп и, если вам по душе более высокие посты, директоров по разработке программного обеспечения. Если вы руководите относительно небольшой группой программистов (состоящей, скажем, из четырех–семи человек), работаете в мелкой или средней компании, заняты разработкой нескольких проектов одновременно — значит, вы выбрали издание правильно. Если же вы, скажем, собираетесь за 12 месяцев построить очередную глобальную систему бронирования авиабилетов, а в вашем подчинении 100 программистов, вероятно, эта книга не будет соответствовать вашему размаху. В таком случае вам лучше защитить две магистерские диссертации: по управлению проектами и по психологии. Удачи.

Если вы руководите программистами — именно руководите — и чувствуете, что лидерство подменяется простым манипулированием проектами и людьми, вам нужна помощь. Моя книга вам поможет.

Быть может, вы — опытный менеджер, желающий пересмотреть свои принципы лидерства. Тогда эта книга опять же для вас.

Не исключаю, что вы наконец-таки получили повышение, на которое давно рассчитывали или которого, наоборот, опасались. Итак, многолетние занятия по написанию интеллектуального кода и конструированию выдающихся программ принесли свои плоды. Начальство посчитало вас подходящим кандидатом на роль руководителя программистов. Вероятно, в вас несколько меньше лунатизма, чем в коллегах. Маловероятно, но все же возможно, что вы привыкли носить сорочки с воротничками, и это обстоятельство сыграло вам на руку. Или кто-то уволился. Как бы там ни было, добро пожаловать в стройные ряды руководителей групп программистов. Моя книга окажется в такой ситуации весьма кстати.

Вне зависимости от возраста, пола и социального статуса эта книга поможет вам укрепить свои позиции в роли лидера программистов. Она довольно компактна, и материал достаточно легко укладывается в голове. Стоя в книжном магазине и раздумывая, что же купить, задайте себе один простой вопрос: «Нужно ли мне совершенствовать свои лидерские навыки?» Полагаю, вы ответите: «Да», — а значит, моя книга окажется для вас небесполезной.

Стиль и позиция

Литература по менеджменту изобилует различного рода предложениями, рекомендациями и научно подтвержденными методиками решения задач при помощи других людей. В конце концов, именно в этом суть менеджмента: делегируя задания подчиненным, вы должны обеспечить мотивацию и проконтролировать (в лучшем смысле слова) выполнение этих заданий ради достижения общей для всей компании цели. Эта книга — отнюдь не очередной фолиант по дисциплине «менеджмент». Скорее, это выраженная в словах концепция, согласно которой для создания первоклассного программного обеспечения требуется мастерство, а оно нарабатывается преимущественно с опытом. Именно из опыта — от этого верного и преданного учителя — я почерпнул те идеи, которые здесь излагаю.

Несомненно, что в моей работе фигурируют принципы, сформулированные в свое время в академической среде. Поскольку, скорее всего, чтением вы занимаетесь в перерывах между телефонными звонками и визитами в соседние комнаты к подчиненным программистам, я постарался излагать материал кратко, в позитивном тоне и с юмором — чтобы хоть как-то развлечь вас посреди рабочего дня[1].

В большинстве случаев я употребляю местоимения в мужском роде. Впрочем, примечание в начале главы 9, надеюсь, прояснит для вас мою позицию в этом деликатном вопросе. Никаких предпочтений я здесь не высказываю — мне кажется, лучше вникать в суть, чем заострять внимание на таких вещах.

В этом своем труде я концентрируюсь на двух основных областях руководства программистами: людях и процессах. Из них люди, разумеется, важнее. При наличии хорошей команды разрабатывать прекрасные программные продукты более чем возможно — даже несмотря на недостатки бизнес-требований. А стоит только подогнать требования, и в сотрудничестве с высококлассными специалистами вы сможете докодироваться хоть до Луны!

[1] Мои шутки, по большей части размещаются в сносках, так что они не будут вас отрывать от основной мысли. Ну вот, например: что значит «обратная совместимость»? Это значит, что новая операционная система с тем же успехом, что и старая, может грохнуть все ваши программы.

Глава 1

Как привыкнуть к роли руководителя

Приступая к новым для нас обязанностям, мы всегда, с одной стороны, питаем большие надежды, а с другой — испытываем определенный страх неудачи. Как сформировавшийся программист вы, конечно, успели приобрести опыт участия в проектах и работы в компаниях. Теперь, получив должность руководителя группы программистов, вы столкнулись с новой и, наверное, слегка обескураживающей задачей. Чтобы преуспеть в неизвестной доселе роли в процессе разработки программного обеспечения, вы должны как можно быстрее пройти путь от программиста до руководителя. При этом вам придется приспособиться к новым общественным условиям и новым механизмам взаимодействия — как с людьми, так и с процессами.

Как известно, совершить рывок чистой физиологии к одухотворенному состоянию человеку помогла адаптация — ведущая сила биологической эволюции. Этот процесс продолжался многие миллионы лет, но результат налицо — люди говорят на разных языках и оперируют абстрактными понятиями вроде компьютерных программ. Так как же мы до этого докатились? Вообще-то этот вопрос лучше задать биологу, но из содержания этой книги вы убедитесь, что адаптация значительно упрощает задачи, стоящие перед руководителями программистов.

Речь в этой главе пойдет о людях. Мы поговорим о самом что ни на есть человеческом занятии — написании кода. В частности, мы разберемся в психологии людей, посвящающих себя этому замечательному занятию. Чем лучше вы знаете людей, которыми вам предстоит руководить, тем выше шансы на успех. Принципов и методик руководства программистами сегодня развелось великое множество. Каждое поколение руководителей, естественно, отталкивается от собственного опыта и устраивает свою деятельность исходя из известных стилей руководства и менеджмента. Лично я принадлежу к тому поколению, которое привыкло орудовать логарифмическими линейками и перфокартами, и, конечно, это обстоя-

тельство накладывает некоторый отпечаток на мои рассуждения. Тем не менее, проработав много лет с программистами значительно моложе себя, я понял, что подходы, свойственные моему поколению, отнюдь не уникальны. Не раз сталкиваясь с трудными задачами руководителя, я старался привыкать к новым направлениям бизнеса, к технологическим изменениям, да и просто боролся с собственным упрямством. Этим своим опытом я намерен поделиться с вами, и очень хочется надеяться, что вместе мы проведем замечательное исследование этого вопроса.

Правда ли, что настоящие руководители ходят в черном?

Некоторые — ходят. Иные даже носят на голове хвосты (хотя это, конечно, зависит от того, сколько у них осталось волос). Так они пытаются повысить свой авторитет в глазах подчиненных им придурков или «ботаников» — выберите тот эпитет, который вам больше нравится. Вполне возможно, вам не нравятся оба варианта, а себя вы ощущаете руководителем современного типа, организующим подобных вам специалистов, которые искренне считают программирование пищей для ума. Что я хочу сказать? Стереотипы (в том числе упомянутые в заголовке раздела) не следует воспринимать слишком серьезно. О чем действительно стоит задуматься, так это о личных взаимоотношениях с программистами и своем месте среди них. Став руководителем, вы уже не имеете права вести себя так, как вели, будучи одним из них. Кроме того, ваше положение шефа в процессе разработки программных продуктов иногда оказывается очень кстати. Как кто-то когда-то сказал: «Дайте мне длинный рычаг, и я поверну Землю». Руководство — это как раз такой рычаг.

Вполне допускаю, что мои пассажи насчет несущественности имиджа вас не убедили. Знаете, я сам довольно долго подходил к мысли о том, что мои внешние атрибуты не обязательно отражают мою внутреннюю сущность. Мне до сих пор нравится стиль «ботаника», но теперь я знаю, что для руководства моей группой этого совершенно недостаточно. Иной руководитель чуть ли не полностью посвящает себя убеждению своих подчиненных в том, что он до сих пор один из них. Но так ли это важно? Вспомните фильм «Сеть» (The Net). Сетевые приятели его главной героини Анжелы (в исполнении Сандры Баллок) признали ее единомышленницей и с готовностью приняли в свой странноватый круг общения. Только вот в конце концов выяснилось, что ничего особо замечательного в этом кругу нет. Отсюда урок: образ человека не может быть поверхностным. Что действительно имеет значение, так это характер. Почему стандартные методики менеджмента так часто на практике оказываются несостоятельными? Да просто потому, что их последователи, вместо того чтобы выработать собственный индивидуальный стиль, забивают методиками свои головы и действуют по ним как по шаблону.

Насколько важно быть крутым?

Итак, если мы заговорили в таком серьезном тоне, стоит ли все-таки постоянно ходить в черном и эксплуатировать стереотипы, которые, по вашему мнению, определяют уровень руководителя программистов? Чтобы оценить, насколько вы крутой, рекомендую пройти тест (см. ниже). Кстати, имейте в виду,

что некоторые предпочитают термину «крутой» слово «нинджитсу»[1], но я придерживаюсь традиционных понятий.

Не забывайте, что моя книга предполагает некоторую работу над собой, — так что не слишком расслабляйтесь. Неожиданные проверки не являются прерогативой старых и противных университетских профессоров — в реальной жизни они подстерегают нас постоянно.

НАСКОЛЬКО ВЫ КРУТОЙ?

Выберите один или несколько вариантов ответов на следующие вопросы.

1. «Хакер» — это тот, кто:
   а) вырубает мебель топором;
   б) не теоретизирует, а увлеченно программирует;
   в) удовлетворяет свои интеллектуальные амбиции, творчески преодолевая или обходя препятствия;
   г) руководствуясь злым умыслом, пытается похитить секретную информацию;
   д) персонаж, сыгранный Анжелиной Джоли.
2. «Взломщик» — это:
   а) человек, который взламывает системы безопасности компьютеров;
   б) белый парень с Юга, вроде вашего автора;
   в) нечто, еще более неуловимое, чем cookie (см. вопрос 6);
   г) латентный хакер.
3. «Фрикинг» — это:
   а) искусство и техника взлома телефонных сетей;
   б) старый «ботаник», строящий из себя крутого.
4. «Пинг» — это:
   а) отправка интернет-пакетов;
   б) писк ультразвукового прибора;
   в) начальная часть названия игры «пинг-понг»;
   г) много счастья, и все сразу.
5. «Червь» — это:
   а) оптический диск с однократной записью и многократным считыванием;
   б) программа-вирус, разрушающая данные в памяти или на диске;
   в) двусторонне симметричное беспозвоночное.
6. «Cookie» — это:
   а) маркер доступа, который передают друг другу взаимодействующие программы;
   б) нечто, ставшее известным благодаря Амосу;
   в) нечто, в чем хранятся (а иногда — анализируются) данные о поведении пользователей при посещении ими веб-страниц.

[1] Вы ведь знаете, что такое «ниндзя», правда? Это слово обозначает исключительность. Получается своего рода «черный пояс в программировании».

Ну как, справились? Однажды мне пришлось читать лекцию по компьютерной безопасности студентам, имевшим к программированию весьма отдаленное отношение. В тот момент я преследовал единственную цель — составить характеристику людей, которые, во-первых, занимаются хакерством, во-вторых, защищают компьютерные системы от потенциальных угроз. Они не слишком хорошо справились с заданием — держу пари, у вас получилось значительно лучше. Дело в том, что все варианты ответов на все вопросы правильны[1] — ну разве что вариант б ответа на вопрос 3 я выдумал. Но на самом деле этот тест успешно подтверждает то обстоятельство, что традиционно программистов приписывают к определенной субкультуре. Иногда ее называют (да не обидятся на меня специалисты) «хакерской» культурой (хотите убедиться? взгляните на варианты ответов на вопрос 1 еще раз — они довольно забавны, не правда ли?). Сегодня стереотипы, связанные с хакерством, понемногу уходят. Даже начинающий программист в сегодняшних условиях — это, как правило, выпускник колледжа или университета, да еще к тому же обладатель магистерского диплома по экономике и управлению. В то же время в каждой компании своя культура, и группа, которой вы руководите, — не исключение. Она в целом не менее уникальна, чем каждый из работающих в ней специалистов. Вне зависимости от определения культуры, она в любом случае совпадает с контекстом ваших обязанностей как руководителя программистов. Стоит хотя бы немного разобраться в культуре ваших подчиненных (в частности, в том, как они мыслят и выстраивают свое взаимодействие), и качество ваших отношений, равно как и эффективность исполняемых вами руководящих функций, сразу возрастет. Так что, если очень хочется, вы, конечно, можете носить черную футболку с таинственным посланием на груди, но имейте в виду, что, подстраиваясь под стереотип руководителя и всемерно подкрепляя его собственным примером, вы сознательно выбираете далеко не самый эффективный стиль управления. Значительно более эффективным механизмам руководства как раз и посвящена эта глава.

Стереотипы, связанные с хакерством, понемногу уходят. Даже начинающий программист в сегодняшних условиях — это, как правило, выпускник колледжа или университета, да еще к тому же обладатель магистерского диплома по экономике и управлению.

Мало быть крутым — смотри в оба!

Конечно, если вы сами отъявленный хакер, проблем по части общения с программистами у вас не возникнет. Тем не менее возьму на себя смелость вас предостеречь: становясь руководителями, даже самые крутые программисты не всегда идеально справляются со своими новыми обязанностями. У них возникает непреодолимое желание самим работать над проектами, которые, по идее, нужно делегировать подчиненным. Они тратят уйму времени на обзоры кода, хотя на это хватит и часа. Они пытаются вылизать все комментарии и отступы. Иногда они бросают попытки объяснить окружающим, что они от этих окружающих хотят, и пытаются все

[1] Большинство этих терминов разъясняются в издании The New Hacker's Dictionary, Third Edition, by Eric S. Raymond (The MIT Press, 1998).

делать сами. Я хочу, чтобы вы меня правильно поняли: вы действительно должны держать в голове как можно больше подробностей, касающихся разработки кода, за который несете ответственность, но также вы должны уметь мыслить глобально, чего, к сожалению, программисты-менеджеры зачастую делать не умеют.

Вы должны держать в голове как можно больше подробностей, касающихся разработки кода, за который несете ответственность, но также вы должны уметь мыслить глобально, чего, к сожалению, программисты-менеджеры зачастую делать не умеют.

Бывает и другая крайность. Некоторые менеджеры днем руководят, а ночью пишут код, — как вы понимаете, ни на что другое у них не остается времени. В таком случае кодирование воспринимается как главная ценность в жизни, а руководство — как неприятная обязанность. Такая схема, в принципе, срабатывает, но лишь до тех пор, пока не пропадает всякое желание работать. С моей точки зрения, любой профессиональный руководитель программистов обязан лелеять в себе страсть к работе, не давать этой страсти исчезнуть. В этой и нескольких следующих главах мы рассмотрим ряд приемов, к которым менеджерам рекомендуется обращаться, чтобы гармонизировать свою работу и в то же время не потерять желания трудиться. Один из таких приемов, позволяющий выделять время на выполнение руководящих функций, — делегирование. Поскольку делегирование есть краеугольный камень всякой руководящей деятельности, ему полностью посвящена глава 8. Пока что вы должны четко уяснить себе, что делегирование невозможно без доверия к своим подчиненным. Для того чтобы построить доверительные отношения, требуется некоторое время, но без них деятельность руководителя обречена на провал. Не следует также забывать, что доверие предполагает взаимность. Мне очень хочется, чтобы, прочитав эту главу, вы научились доверять своему интуитивному представлению о людях и путем некоторого анализа интеллектуальных способностей и личностных характеристик своих программистов смогли лучше в них разбираться.

Как руководить чокнутыми, чудаковатыми, странными и обычными программистами

Не хотелось бы говорить об управлении программистами исключительно в ироническом тоне — хотя надо заметить, что этот вид деятельности кто-то очень удачно сравнил с выпасом котов, имея в виду, конечно, творческий характер людей, занятых написанием кода. Проблема в том, что управлять этими иногда беспокойными, неизменно нужными и, как правило, очень интересными сотрудниками крайне трудно. Чем лучше вы будете их знать, тем совершеннее станет ваш стиль руководства.

Если вы программист-эстет, то выражение «быть на короткой ноге с кодом» вам, конечно, известно, — код в таком случае оказывается чуть ли не второй натурой программиста. Элен Алман (Ellen Ullman) в своей книге «Close to the Machine» выражается по этому поводу следующим образом.

«Один из моих знакомых руководителей проектами однажды сравнил процесс управления программистами с выпасом котов. Он хотел сказать, что песики,

преданно заглядывающие в глаза, нам совершенно не нужны. Хорошего программиста нужно ценить вместе со всеми его странностями. С другой стороны, всех этих хороших программистов нужно каким-то образом заставлять двигаться в одном направлении»[1].

Вот это «одно направление» отражает задачу, которая стоит перед каждым руководителем проекта. Но ведь двух одинаковых программистов не существует, и поэтому для каждой группы специалистов нужно выбрать индивидуальный стиль руководства. Управлять программистами невозможно, если в них не разбираться. Поэтому в следующем разделе я привожу перечень характерных «типов» программистов и для каждого из них обозначаю отличительные характеристики. Скорее всего, в них вы узнаете кого-то из своих подчиненных. Поскольку наша книга о котах, типы я называю «породами».

Какие бывают породы программистов

На что похож типичный программист? Можно ли построить шаблон программиста, хотя бы для того, чтобы понять мотивы его поведения? Может быть. В психологической науке существуют так называемые тесты оценки личности. Здесь тот же принцип — достаточно разобрать отличительные черты программистов в отдельности, и вы поймете, что многие из этих характеристик (даже если они на первый взгляд кажутся взаимоисключающими) могут существовать в одном лице. Я разделяю породы на три категории: распространенные, редкие и дворовые. *Распространенные* породы встречаются чаще всего. *Редкие* породы встречаются не так часто, но иногда с ними все-таки приходится сталкиваться. *Дворовые* породы, как и следует из их названия, приносят не слишком много пользы, но знать о них нужно, хотя бы в силу того, что они существуют. Если регулярно помогать дворнягам повышать уровень знаний и, соответственно, преодолевать слабости, которые они по определению привносят в процесс кодирования, они могут стать очень достойными исполнителями.

Как я уже говорил, любой отдельно взятый человек может заключать в себе характеристики сразу нескольких пород. Конечно, разобраться в таких людях труднее, но это того стоит. У программистов на редкость богатое «наполнение». Различия между породами и индивидуальные стили, характерные для каждой из них, как мне кажется, нужно смаковать. Вполне возможно, что в следующий раз вы столкнетесь с этими характеристиками, невзначай взглянув в зеркало.

Распространенные породы

Ниже я перечисляю распространенные породы и их характеристики.

Архитектор[2]**.** Большинство руководителей обожают этот тип программистов — и, действительно, любой такой деятель окажется ценным приобретением для вашей команды. В основном архитекторы концентрируются на общей структуре кода. Они мыслят объектами, а их лучший друг — лист белой бумаги. Посвящая себя без остатка решению бизнес-задач, они строят абстракции, проводят

---

[1] Ellen Ullman, Close to the Machine (San Francisco: City Lights Books, 1997), p. 20.

[2] Термином «архитектор» я в данном случае обозначаю один из личностных типов программистов и совершенно не имею в виду полноценного программного архитектора. О том, какую роль играет архитектура в общей схеме разработки, говорится в главе 6.

анализ систем, после чего переходят к кодированию конкретных решений. Слов нет — все это очень важные элементы программирования, но для комплексного выполнения задач их еще не достаточно. Зачастую в высшей степени разумные замыслы архитектора воплощаются в настолько общем и непонятном коде, что людей, могущих разобраться в нем и продолжить начинание, просто не находится. Особи, способные генерировать удачную идею в голове (а лучше в Visio), а затем выполнить ее полноценную конкретизацию в коде, становясь, таким образом, единственными участниками процесса, встречаются очень редко. Недостаток архитекторов в том, что их код часто служит только одному хозяину, а исполнять чужие команды категорически отказывается[1]. Некоторые архитекторы очень любят набросать структуру кода, с тем чтобы впоследствии передать его на растерзание программистам более «низкой» квалификации. Иногда в коде, написанном архитекторами, встречаются весьма странные конструкции — например, окна с сообщениями о системных прерываниях из-за ошибок, появляющиеся по той лишь причине, что код предполагалось исполнять в виде библиотеки DLL на сервере.

Конструктивист. Конструктивисты получают удовольствие от процесса написания кода и его результата. Стратегическим планированием они себя утруждают не всегда, но факт в том, что с написанием кода они справляются быстро, причем в большинстве случаев ошибок в нем не обнаруживается даже на этапе альфа-тестирования. Код конструктивисты пишут по наитию, а потому их логика не всегда понятна. У некоторых конструктивистов все в порядке и с интуицией, и со стратегическим планированием, поэтому код выступает естественным продолжением хода их мыслей. Но стоит попросить конструктивиста составить документацию, он обязательно ответит, что код самодокументируемый. Впрочем, если на него немного надавить и дать понять, что без документации никуда не деться, он, вероятно, согласится ее составить — и сделает это качественно. Кто спорит — код должен быть самодокументируемым, но следует иметь в виду, что основное внимание программисты этого типа уделяют процессу создания кода, поэтому остальное для них не так уж важно. Количеству сборок, которое конструктивист выдает за день, позавидует даже Microsoft. Соответственно, их код обычно отличается надежностью. Однако же по мере разбухания (а этот процесс неизбежен) надежность улетучивается, а конструктивист начинает судорожно искать новые, «заплаточные» решения — ведь для него очень важно видеть результат и пребывать в уверенности в том, что он справился с поставленной задачей. Конструктивист в сочетании с архитектором имеют все шансы стать прекрасной командой. Если же вы умудритесь отыскать конструктивиста и архитектора в одном лице, считайте, что львиная доля кадровых проблем решена.

Художник. На самом деле, искусства в написании кода не меньше, чем науки, — не зря же университеты часто сводят оба направления в одной структуре и называют ее как-нибудь вроде «факультета свободных искусств и наук». Не будь в программировании художественного аспекта, может быть, оно приносило бы

[1] А ведь это очень важно — согласно авторитетным оценкам, по меньшей мере 70 % стоимости программного обеспечения приходится на сопровождение. См. William H. Brown et al, AntiPatterns: Refactotoring Software, Architectures, and Projects in Crisis (New York: John Wiley & Sons, 1998), p. 121.

нам гораздо меньше морального удовлетворения. Художник как тип программиста сконцентрирован на процессе создания кода — переносе коммерческих требований на программные конструкции и искусном сведении объектов пользовательского интерфейса в одну изящную структуру. Работая с компонентами без видимого интерфейса, художники обнаруживают тенденцию к правильной и логичной организации. Недостаток художника в том, что очень часто он затягивает кодирование, пытаясь выяснить, сколько знаков равенства можно установить в одной строке, не нарушив при этом правильность результата булева оператора. С другой стороны, если программист не культивирует в себе художника, результаты его деятельности зачастую отрываются от реальности, теряют «изюминку». Стоит отнять у художника все его отличительные качества, и в результате получится мина замедленного действия, которая обязательно взорвется под пальцами пользователей. Разделяя некоторые характеристики конструктивистов и архитекторов, художники активно претендуют на собственный стиль.

Инженер. Инженеры вам понравятся. Эти ребята имеют обыкновение скупать все возможные средства сторонних производителей, писать десятки COM-объектов и сводить их воедино, так что они прекрасно работают в версии 1. Присущая им тяга к усложнению проявляется лишь тогда, когда речь заходит о версии 1.1. Программирование часто приравнивают к инженерии программных средств — и, действительно, многие стороны нашей профессии подчиняются этой методологии. Но давать инженерам карт-бланш нельзя. В программных продуктах, выстроенных инженерными методами, нет ничего предосудительного — в конце концов, согласно классическому определению, инженерия есть научные принципы, задействованные при решении программных задач. Нам нужны программисты, которые не боятся сложностей, но те из них, которые любят усложнять все и вся, представляют серьезную опасность. Поймите меня правильно: я совершенно не собираюсь бросать камень в огород инженеров. В конце концов, я сам многие годы трудился над аппаратным обеспечением компьютеров. Но аппаратная направленность иногда входит в противоречие с теми аспектами программного обеспечения, благодаря которым оно становится программируемым (то есть гибким и многократно используемым). Любое аппаратное устройство обычно служит одной, четко определенной цели, а для программного обеспечения такой подход неприемлем.

Ученый. Ученые — это мальчики и девочки, которые считают себя последователями Бэббиджа и Тьюринга. Никогда в жизни они не вставят в код инструкцию GoTo. Отодвигая художественную составляющую программирования на второй план, они делают все в соответствии с фундаментальными принципами компьютерных наук. И как раз в этом обычно и заключается проблема. В то время как они одержимы безупречностью своих трудов, ваша главная забота как руководителя состоит в том, чтобы разработать доброкачественный продукт и сдать его к установленному сроку. Программисты такого типа на самом деле очень полезны, и когда речь заходит об особо трудных задачах кодирования, их идеям нет цены. Вы просто должны следить за тем, чтобы их педантичность не перевесила практические соображения. У инженеров и ученых есть одна общая черта — те и другие очень любят все усложнять. Иногда даже кажется, что они все поклоняются богу сложности (и даже приносят ему жертвы!).

Отдавая должную оценку глубочайшим познаниям ученых и по мере необходимости обращаясь к ним, руководитель не должен допускать их полновластия в вопросах написания кода — иначе они сделают его слишком сложным.

Лихач. Лихачи — это те товарищи, которые делают все быстро. Забывая о комментариях, отступах и соглашениях об именовании переменных, они, тем не менее, умудряются достигать результата очень оперативно — и, что самое замечательное, вплоть до первой неперехваченной ошибки их продукты вполне успешно работают. Иногда такое поведение характерно для молодых программистов, горящих желанием впечатлить вас, — они опрометчиво считают, что оперативность в достижении результата в полной мере соответствует вашим ожиданиям. Признайтесь: мы часто сами выстраиваем у них столь ложное представление, а значит, веди мы себя по-другому, никаких лихачей не было бы. Наши собственные начальники устраивают совещания, на которых устанавливают контрольные сроки, а потом сообщают их нам. Как мы добьемся выполнения поставленных временных задач — это уже наша проблема. Вспомните, как часто идут разговоры о бессмысленности установления крайних сроков кодирования до окончательного выяснения всех требований! Так вот, *вам придется к этому привыкнуть*. К сожалению, такова реальность — пользователи и рыночные соображения часто принуждают нас сперва давать обещания, а потом уже приступать к планированию. Именно по этой причине вы читаете мою книгу — вам нужны советы по поводу того, как выжить в динамичном, жестоком и суровом мире разработки программных средств.

Редкие породы

Далее я привожу список редких пород и их отличительных черт.

Волшебник[1]**.** Каким-то загадочным образом те, кого я называю волшебниками, регулярно решают самые трудные задачи программирования, причем идут такими путями, которые раньше никому и в голову не приходили. Более того — волшебники делают все это вовремя, и иногда у них получаются вполне доступные для понимания программы, которые даже можно сопровождать. Немного волшебства в нашем деле не помешает. Но стоит распустить подобным деятелям руки, и вскоре вы превратитесь из здравомыслящего руководителя работоспособной группы программистов в обычного подмастерье. Кроме того, если вы будете слишком полагаться на волшебника, в один прекрасный день он вас разочарует — в конце концов, постоянно творить чудеса никому еще не удавалось.

Минималист. Несмотря на удивительно скромный объем кода, производимого минималистами, код обычно оказывается очень функциональным. Каждая процедура умещается в редакторе кода на одном экране. Объекты стройны, выстроены четко и недвусмысленно сообщают о своем назначении. Звучит неплохо, не правда ли? В общем, да, только стоило бы учитывать мотивы такого поведения. Ведь иногда они кроются в том, что человеку хочется побыстрее разобраться с текущим проектом и перейти к следующему, который его больше захватывает. Иногда (кстати говоря, эта характеристика распространяется и на

[1] Некоторые предпочитают называть программиста этого типа «гуру» или «спецом». А мне больше нравится «волшебник».

архитекторов) минималисты, решив поставленную задачу, быстро теряют к ней всякий интерес, и уж, конечно, при обнаружении в ходе альфа-тестирования каких-либо проблем выказывают устойчивое нежелание их исправлять. Иногда минималисты капризны и очень придирчиво выбирают область приложения своих сил. С сопровождением кода дела у них обстоят хуже всех.

Аналогист. Ну ладно, ладно — слово «аналогист» я взял с потолка. Только не подумайте, что это медсестра, которая делает наркоз перед операцией. Нет — это программист, который не слишком силен в абстракциях, но прекрасно справляется с аналогиями. Во время проектных совещаний аналогисты, постоянно выдумывающие все новые и новые аналогии, способны свести с ума любого. Но при этом нельзя не признать, что, как правило, они очень быстро схватывают суть задачи и в результате создают удобный (в том числе и для сопровождения) код. У некоторых аналогистов есть любимые аналогии, которые они норовят применить ко всем без исключения проблемам разработки программных продуктов. Они воображают компоненты маховиками, а успешно справившись со своей задачей, хвалятся тем, что их код «воспламеняется во всех цилиндрах». Их аналогии всегда привязаны к осязаемым объектам, поскольку, как я уже говорил, с абстрагированием дела у них обстоят неважно. Ну, в общем, вы меня поняли. Посадите аналогиста вместе с архитектором, и если они сразу друг друга не прикончат, скорее всего, у них получится превосходный продукт. Правда, поскольку аналогисты не дружат с абстракцией, создавать объекты с четкими межуровневыми интерфейсами у них получается не всегда. Дело в том, что возможность создания в достаточной мере абстрактного интерфейса объекта — это одно из величайших преимуществ объектно-ориентированного программирования, и поэтому конкретное мышление иногда мешает успешно справляться с поставленными задачами.

Трюкач. Трюкачи слишком увлекаются разными технологическими трюками. Они постоянно осваивают разные новинки, но результат от этого не улучшается. По правде говоря, всех нас в той или иной степени привлекают забавные технологические приемы. Я вот, например, помню мой первый компьютер. Он был аналоговым, и, поворачивая диски, я переключал ветви в предустановленном аппаратном алгоритме. Эта штука была похожа на гипертрофированную логарифмическую линейку. В общем, я до сих пор люблю забавляться со всякими высокотехнологичными штуковинами. Если вам приходится работать с трюкачами, попытайтесь направить их увлечение игрушками на решение их первоочередной задачи, а именно на производство бизнес-решений. Если им удалось втиснуть на экран, который, как предполагается, будет отображаться с разрешением 800 × 600, 30 разных элементов пользовательского интерфейса, это еще совершенно не означает, что они решили свою задачу в соответствии с реальными потребностями пользователей[1]. Трюкачи, при всех их познаниях в технологии, часто не могут усвоить конечное назначение программы. Полагая, что их функции ограничиваются забавами с разными инструментальными средствами, они отказываются учитывать те аспекты программирования, благодаря которым мы не затрачиваем на сопровождение титанических усилий.

---

[1] Вы тоже ненавидите пользователей?! Представляете, как было бы здорово писать программы только для программистов?

Дворняги

Остановимся теперь на дворовых породах и их отличительных чертах.

Разгильдяй. Что сказать о разгильдяях? Некоторые люди небрежны, и это проявляется в коде, который они создают. Они не обращают внимания на такие мелочи, как правильное написание имен переменных и правила венгерской нотации. Зачастую качественно выполнять свои обязанности им мешают проблемы личного плана. Тому, как пишется эффективный код, их нужно учить. Они любят начать с одного стиля, а через процедуру-другую перейти к новому. Читать их код очень утомительно — иногда поздними ночами его приходится переписывать, поскольку иначе есть риск не успеть к сроку сдачи проекта. Если вы не справились с задачей по их вине, прошу вас: отнеситесь к ним снисходительно. В конце концов, они просто отъявленные разгильдяи, которых лучше всего пересадить в отдел бета-тестирования. Хотя нет — так вы просто заморозите проблему, в итоге она все равно может проявиться. Если разгильдяй действительно любит писать код, при условии, что вы уделяете ему достаточно внимания, он имеет шансы реабилитироваться. Всех, кому это не удается, нужно просто пнуть под зад или познакомить с консультантом по трудоустройству.

Тормоз. Тормоз — это программист, который не знает, с чего начать. Он постоянно ищет спецификацию (или ожидает, пока ему дадут), отчаянно надеясь, что она станет для него отправной точкой. Нерешительность в чем-то хороша, поскольку в некоторых случаях она повышает качество кода. Однако иной раз она свидетельствует о низкой квалификации программиста, который не хочет лишних ошибок на этапе прогона. Предоставьте этим ущербным образец кода, чтобы они могли разобраться, с чего начинать, и выбрать стиль, которого им нужно будет придерживаться. Нерешительность часто характерна для неопытных программистов, и, воспользовавшись некоторыми воспитательными методами, вы можете наставить их на путь истинный. Кроме того, нерешительностью иногда страдают программисты, у которых по тем или иным причинам не слишком впечатляющий послужной список. Ну, скажем, в прошлый раз их результаты разнесли в пух и прах, а теперь они хотят исправиться, но очень боятся наступить на те же грабли. Действительно, низкая самооценка часто проявляется в форме нерешительности. С такими типажами нужно проявлять терпение. Помогите тормозу регулярно добиваться небольших успехов, и тогда все наладится. Наставничество (лучший, кстати говоря, способ перевоспитания нерешительного программиста) рассматривается в главе 8. В ней я утверждаю, что роль наставника — это одна из основных ролей руководителя программистов, которая при должных вложениях обязательно окупится.

Любитель. Любители очень хотят стать настоящими программистами. Тщательно изучив какой-нибудь инструмент написания макрокоманд, они возводят себя в ранг хакеров. Единственная причина, по которой они бросают уютные места в отделах поддержки пользователей и тестирования, заключается в том, что, по их мнению, быть программистом — это очень круто. Да, мы, действительно, крутые, но, по большому счету, это лишь побочное следствие нашей основной деятельности. Любителям не хватает образования, но по мере их обучения вы должны пристально за ними следить и лишь при условии определенных достижений с их стороны поручать им работу над критически важными приложениями. Узнав на собственном опыте, как трудно заниматься программированием и какое

серьезное внимание к деталям требуется от программистов, любители часто разочаровываются в своем выборе. Они отказываются признавать превосходство объектно-ориентированных методов над процедурной парадигмой — и все потому, что нужное прозрение их еще не посетило. В защиту любителей вспомним замечательное высказывание: «любители построили ковчег, профессионалы построили "Титаник"». На самом деле иногда свежий, незашоренный взгляд начинающего программиста очень помогает нам — старым брюзгливым технарям.

Профан. Программист-профан — это тот, кого называют тупицей. Хуже всего, когда профан не догадывается о своей тупости. Остерегайтесь таких людей. Иногда они могут достаться вам в наследство от предыдущих руководителей, но сами, я вас прошу, никогда их не нанимайте. У меня нет никаких предубеждений относительно умственно ущербных людей, но я твердо уверен, что в профессии, требующей постоянного самосовершенствования и обучения, таким не место. Если человек невежда, но хочет стать лучше, — дайте ему шанс. Отправьте его, например, в отдел тестирования — иногда не отличающиеся выдающимися умственными способностями пользователи находят себя в отлове жучков[1]. Еще одно соображение насчет глупости: на самом деле все мы постоянно страдаем от несовершенства того, что находится между клавиатурой и стулом. Но, в конце концов, если бы для написания кода не требовались мозги, этим занимались бы все без разбору, так ведь? Я советую не путать невежество с глупостью. Невежество исправимо, а с глупостью лучше просто не связываться. Если вы унаследовали кадры, подобранные не программистом, вполне возможно, что среди ваших подчиненных есть такие типажи. Руководители, имеющие весьма отдаленное представление о технологии, иногда покупаются на необоснованные заверения бездарных претендентов на место.

Эклектик. Можно сказать, что эклектики просто стряпают программные продукты. Представитель этой породы сочетает в себе качества инженера, разгильдяя и не слишком талантливого художника, причем упомянутые ингредиенты находятся в чудовищной диспропорции. Результат их деятельности представляет собой винегрет из стилей кодирования и подключаемых модулей при невероятной путанице в коде. Все это выглядит довольно привлекательно, но стоит лишь попробовать кусочек, как наступят необратимые последствия. Отправьте такого программисты на кулинарные курсы и обязательно проверьте, не скрывается ли за внешней оболочкой талантливости банальный разгильдяй. В классическом виде эта дворянская порода встречается довольно редко, а упомянул я ее по той причине, что отдельные ее черты проявляются в стилях кодирования самых разных типов программистов. Если они не считают нужным следовать корпоративным стандартам, вам придется посвящать все рабочее время напряженным попыткам выяснить, что же они все-таки имели в виду и как сопровождать их код. Основным средством реабилитации эклектиков служит критика кода (см. главу 6).

Умение обращаться с представителями разных пород

Программисты — это в первую очередь люди. Поэтому в одном человеке могут быть в большей или меньшей степени выражены все перечисленные характеристики. Некоторые из них как будто исключают друг друга, но на самом деле это не так.

---

[1] Лично я предпочитаю термину «жучок» (bug) словосочетания «программная аномалия» (program anomaly) и «открытие недокументированной характеристики» (Undocumented Feature Offering, UFO).

Все люди сотканы из противоречий, и ваши подчиненные — не исключение. От вас как от человека, осуществляющего руководство этими чудесами природы, требуется понимание, умение мотивировать и, прежде всего, мудрость, которая нарабатывается только с опытом. Мнение о программистах нужно составлять по тем граням их характера, которые ярче других сверкают в свете новых начинаний и ослепляющих вспышек проектов, приближающихся к сдаче.

Мнение о программистах нужно составлять по тем граням их характера, которые ярче других сверкают в свете новых начинаний и ослепляющих вспышек проектов, приближающихся к сдаче.

Предположим, у вас есть счастливая возможность набрать сотрудников в свой отдел с «чистого листа». Какие породы сочетаются удачнее? По-моему, лучше всего соблюдать баланс между архитекторами и конструктивистами. Эти две породы привносят в процесс создания программных продуктов наиболее востребованные навыки — первые мыслят стратегически, вторые прекрасно ориентируются в деталях. К этому альянсу время от времени имеет смысл подключать художников. К сожалению, скорее всего, подобрать группу из идеальных кандидатов не удастся. Работать вам придется с тем, что есть. Потому успех вашего взаимодействия с людьми, сочетающими в себе вышеупомянутые характеристики, зависит от вашей проницательности, терпения и умения быть для подчиненных наставником — то есть от трех универсальных качеств руководителя.

Есть еще один тип личности, на который следует обращать особое внимание. Я имею в виду программистов-ковбоев. Этот тип плохо согласуется с перечисленными породами, а описывать его лучше в соответствии с тем мнением, которое ковбой о себе формирует. Итак, программист-ковбой обычно в совершенстве владеет своим ремеслом, но при этом управлять им практически невозможно. Ковбои глубоко убеждены, что могут работать только над теми проектами, над которыми хотят, делать это на собственных условиях, согласуясь исключительно с собственными планами и обращаясь только к подходящим по их мнению средствам. Такой программист — своеобразный волк-одиночка (или, если придерживаться терминологии этой книги, — кот, который гуляет сам по себе). В зависимости от того, что вам нужно, и вашей готовности терпеть своеобразие их личности, ковбои могут творить либо чудеса, либо хлам. С ковбоями надо держать ухо востро: они ни при каких обстоятельствах не станут частью вашей команды. Прибегать к их услугам стоит либо в безвыходных ситуациях, либо если разрабатываемый проект должен радикально отличаться от всех других, а сопровождать его будут сторонние специалисты.

Почему в программистах сочетаются все эти чрезвычайно занимательные личностные характеристики? Мне кажется, связано это с тем, что сам характер деятельности разработчика программного обеспечения привлекает людей совершенно определенного рода. В своем классическом труде «The Mythical Man-Mouth» Фредерик Брукс (Frederick Brooks)[1] утверждает, что наше ремесло приносит людям удовольствие пяти видов.

1. Радость созидания.
2. Радость созидания полезных для других людей продуктов.

[1] Frederick P. Brooks, The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (New York: Addison-Wesley, 1995), p. 230. Это действительно непреходящая классика. В нашей области очень мало книг, которые переиздаются через 25 лет после появления, и это — одна из самых стоящих.

3. Привлекательность процесса упорядочивания головоломных объектов, состоящих из взаимосвязанных динамичных элементов.
4. Радость от постоянного обретения новых знаний и решения нестандартных задач.
5. Интерес к работе с продуктами, созданными исключительно путем приложения интеллектуальных усилий человека, которые, тем не менее, существуют, развиваются и делают совершенно непередаваемые вещи.

Все эти факторы кажутся тем людям, которыми вы руководите, чрезвычайно привлекательными. Разобравшись в их мотивации (да и в своей тоже), вы сможете серьезно усилить свои позиции как руководителя.

КОШАЧЬИ РАЗБОРКИ — СОРЕВНОВАНИЕ ПО ШИПЕНИЮ И ЦАРАПАНИЮ

Проектное совещание превратилось в словесную схватку между Джоном и Кевином. Мы уже перешли к обсуждению регистрации пользователей в системе, а они все еще спорили насчет низкоуровневых подробностей методик конструирования. Дискуссия застопорилась — ведь, несмотря на отсутствие четкого плана обсуждения, все мы были уверены в том, что поговорить есть о чем. Джон и Кевин спорили без перерыва. Дело в том, что Джон (консультант) и Кевин (опытный сотрудник и талантливый программист) руководствовались совершенно разными мотивами и планами относительно этого совещания. Каждый из них считал своим долгом доказать другому, что он умнее, и вопросы проектирования системы их в тот момент не интересовали. А все потому, что Джона назначили руководителем проекта — он занял должность, на которую очень хотел попасть Кевин. Нашего начальника на совещании не было, и ни одного желающего выступить в роли посредника не нашлось.

На следующий день в центре внимания опять оказались Джон и Кевин — они собачились за каждое проектное решение, и все остальные сотрудники отдела предпочитали помалкивать. К концу второго дня мы сумели согласовать значительно меньше вопросов, чем планировались, а то, что нам удалось, оказалось результатом ожесточенной бойни между двумя воинственными котами — Джоном и Кевином. Остальная часть группы чувствовала себя совершенно дезориентированной. Люди начали сомневаться, удастся ли вообще закончить работу над системой. Значительно осложняло ситуацию то обстоятельство, что перед группой поставили задачу сделать систему как можно быстрее, ибо та ее версия, которая в тот момент находилась на рынке, негативно сказывалась на репутации компании.

Этой короткой историей я хотел обозначить трудности, связанные с введением в одну команду консультантов и программистов. Особенно их отношения осложняются в отсутствие начальника. Вы могли бы справедливо поставить под сомнение разумность выбора в качестве руководителя проекта консультанта. Кроме того, как видите, за неимением четкого плана и механизма разрешения противоречий на проектном совещании сотрудники просто убивают время, подрывая тем самым принцип «сначала проектирование — потом кодирование». Развитие межличностных отношений в группе разработчиков подтверждает мою мысль о необходимости психологического анализа подчиненных для повышения качества руководства.

Конец этой истории довольно печален. Проект закрыли, а проектирование и конструирование продукта поручили группе разработчиков из другого отдела. Пропустившему основные «военные действия» руководителю отдела по возвращении пришлось в течение нескольких дней с большими трудностями объяснять начальству, почему после всех обещаний и заверений, предшествовавших началу работу над проектом, разработчики так и не сдвинулись с мертвой точки.

Слава, почет и деньги

Любой сотрудник жаждет признания своей деятельности, хочет ощущать весомость своего вклада в общее дело. Быть оцененными по достоинству стремятся и некомпетентные сотрудники — даже несмотря на то, что их вклад в деятельность компании исключительно негативный. Все мы любим поразглагольствовать насчет того, что создание кода для нас — это уже своего рода награда. Но хотелось бы мне знать, сколько мы проработаем, если нам не будут за это платить. Но даже если будут платить, и платить хорошо, мы в любом случае будем требовать признания со стороны наших товарищей и надеяться, что начальник между делом похлопает нас по плечу. Настоящие кошки предпочитают спокойно дремать в углу, не заботясь о том, кто и как на них смотрит; кошки-программисты в этом отношении, напротив, проявляют некоторый эксгибиционизм. Ярким подтверждением того, что и программистам нужно признание, служит практика встраивания в код своеобразных «пасхальных яиц» (хотя сегодня она считается несколько вульгарной). Вас, начальника, они воспринимают как зрителя, сидящего в переднем ряду, и, конечно же, ожидают аплодисментов. Причем ожидают все — даже те, над которыми впору начинать читать молитву[1].

Расхваливать своих сотрудников — это, конечно, замечательно, но иной раз стоит на секунду остановиться и поразмыслить, отрабатывают ли они те деньги, которые получают. Ведь это входит в вашу задачу как руководителя. Если хотите похвалить человека, сделайте это на виду у всех. Если считаете нужным покритиковать его, не выносите свои оценки на суд публики. Дело не просто в вежливости — эти правила поведения важны в том смысле, что как похвала, так и порицание одного человека на самом деле оказывают грандиозное влияние на всю группу. Поверьте мне, публичное унижение не способствует повышению продуктивности работы группы. Напротив, похвала, высказанная при всех, причем искренне и по заслугам, способна творить чудеса. Не надо кричать о том, что ребята прекрасно поработали, выходя из комнаты для совещаний. Выберите такое время, когда слова благодарности окажут наибольшее воздействие. Четко определитесь с тем, за что вы хвалите человека, и обязательно сделайте так, чтобы сотрудники группы это знали.

Расхваливать своих сотрудников — это, конечно, замечательно, но иной раз стоит на секунду остановиться и поразмыслить, отрабатывают ли они те деньги, которые получают.

И еще немного о поощрении. Кто знает — может быть, у вас у самого такой начальник, который никогда вам слова доброго не скажет. Тогда и вам будет не до комплиментов подчиненным. Роль лидера предполагает стремление к успеху подчиненной вам группы. Когда группа добивается успеха, вы ее хвалите. Кто же хвалит вас? Иногда никто, хотя время от времени вы сами хотите получить от начальника хлопок по плечу. Роль руководителя, которая предусматривает приложение некоторых усилий для улучшения репутации подчиненных, превратится в сущий ад, стоит вам попытаться искать почестей для самого себя. Не стоит

[1] Я имею в виду тех, кому пора готовиться к основательной порке.

забывать, что любой руководитель должен оценивать свои успехи исключительно по тому, насколько эффективно работают его подчиненные.

Если вы выдвинулись в руководители из программистов, задача усложняется, поскольку теперь вам придется принять ответственность за профессиональную судьбу ваших друзей. Но не позволяйте дружбе мешать делам. Лучше используйте ее как средство достижения общей цели. Я вас уверяю, что если вы, в конечном итоге, все вместе добьетесь успеха, ни один из ваших подчиненных не осмелится утверждать, что вы манипулируете дружескими отношениями.

Мотивирование деньгами

Кажется, я уже поднимал денежный вопрос? Лучше было бы его как-то обойти, поскольку, как сказано в Библии, «...ответом всему — монеты»[1]. Согласно последнему статистическому исследованию, почасовая ставка программистов составляет от 30 до 150 долларов, причем большинство из них находится где-то посередине этого спектра. Как определить, стоит ли платить вашим сотрудникам именно столько, сколько вы им платите? Среди факторов, которые нужно учесть, — производительность, опыт, результативность, своевременность исполнения задач, текущая средняя ставка, экономическая конъюнктура и корпоративные стандарты, касающиеся оплаты труда сотрудников в области высоких технологий. Подбирая новых сотрудников и повышая оплату старым, вы должны быть справедливым и бережливым одновременно.

Справедливым и бережливым. Хм, это довольно трудно — можно поддаться соблазну разбрасываться деньгами, как будто они растут на деревьях, опрометчиво полагая, что оплата повышает производительность. На самом деле здесь стоит хорошенько подумать, ведь сегодняшняя роскошь есть завтрашняя потребность. Деньги, подобно власти, оказывают на человека самое что ни на есть деструктивное влияние. И не заставляйте меня цитировать еще одно известное высказывание из Нового Завета[2]. Так все-таки, возвращаясь к теме, как соблюсти баланс в вопросах денежных выплат? Представим себе весы правосудия. На одну чашу положим справедливость, на другую — бережливость. Вес чаши справедливости равен опыту и производительности программиста. Вес чаши бережливости состоит из стандартных коммерческих хитростей, таких как соблюдение баланса доходов и расходов и статистика средней заработной платы программиста. Принимая решение о денежных выплатах, имейте в виду эту аллегорию — она очень действенна.

Но ведь это все теория. А что на практике? Именно из-за несоответствия теории и практики денежный вопрос в деятельности руководителя становится одним из самых сложных. Вам известны принципы, следуя которым вы теоретически должны принимать решения относительно вознаграждения персонала. С другой стороны, существенные коррективы в планирование вносят экономические аспекты — в частности, текущая коммерческая конъюнктура и корпоративная политика. В некоторых компаниях, помимо оклада, сотрудники получают бонусы, которые выписываются в соответствии с личными заслугами каждого.

---

[1] Ну вообще-то в Библии это высказывание приписывается глупцу. Контекст смотрите в Экклезиасте 9:14–19. Только постарайтесь не слишком падать духом, когда будете читать.

[2] См. Новый Завет, 1-е послание к Тимофею 6:10 — любовь, деньги и зло он виртуозно сводит в один силлогизм.

Этот стимул действует лишь тогда, когда сотрудники оказываются в состоянии выполнять то, за что они теоретически заслуживают дополнительного вознаграждения. В принципе, можно ввести практику выплаты ежеквартальных бонусов, но, по моему опыту, это не самый лучший выход — однажды начав их выплачивать, вы обязательно столкнетесь с тем, что сотрудники будут на них надеяться. Касательно денежного вопроса вам лучше проконсультироваться со своим начальником — скорее всего, вместе вы сможете разработать оптимальный для своей команды план. Если решения о вознаграждении принимаете только вы, действуйте так, как считаете нужным, и не забывайте правило справедливости и бережливости.

Уровень мышления[1]

Ну что, вам интересно? Вы заинтригованы тем, что будет дальше? Думаю, что нет. Скорее всего, вы пребываете в полной уверенности, что дальше я разражусь мириадами советов о том, каких действий лучше избегать, а на каких делать упор, и все эти сведения будут представлены в виде маркированных списков. Действительно, в последующих главах таких списков будет немало, но в данный момент я хочу обратить внимание на то, как важно в вашей новой роли думать. Поэтому с перечислением рекомендуемых и не рекомендуемых приемов руководства пока что повременим. Возможно, необходимость думать — это самая сложная обязанность менеджера. Тем не менее это абсолютно необходимое условие нашего успеха. Как пишет в книге «Dynamics of Software Development» Джим Маккарти (Jim McCarthy):

> «Основная задача руководителей процесса разработки программных средств состоит в том, чтобы аккумулировать как можно больше интеллектуальных ресурсов и направить их на создание конечного продукта»[2].

Стоя в душе — думайте. Катаясь на велосипеде, прогуливаясь по парку, выделывая невообразимые трюки на роликах — думайте. Сталкиваясь с дилеммами, которые обусловлены принятыми проектными решениями, — думайте. Думать значительно полезнее, чем смотреть телевизор или бесцельно бродить по Сети, — пусть даже там 500 каналов, но на самом деле на них ничего не происходит, и то, что они как будто избавляют человека от необходимости мыслить, совершенно не здорово. Думайте напряженно, до изнеможения, а когда не останется сил — начинайте заново. Результат вас удивит.

Ну а теперь подумаем о том, как справляться с некоторыми типичными ситуациями.

Предположим, в вашем подчинении кот породы минималист, но при этом весьма профессиональный. Вы хотите поручить ему модернизацию продукта, который писал кто-то другой, но доработать который в контексте текущих коммерческих задач совершенно необходимо. Взглянув мельком на код, который вы собираетесь вменить ему в обязанность, он говорит: «Нет, это слишком сложно — код нужно переписать». Естественно, речь идет о коде, который писался два года и который уже сейчас приносит компании неплохие деньги. Ситуация осложняется тем,

---

[1] Игра слов «шлюзовой уровень» — thunking layer, «уровень мышления» — thinking layer. — *Примеч. перев.*

[2] Jim McCarthy, Dynamics of Software Development (Redmond, WA: Microsoft Press, 1995), p. 5.

что человек, который этот код написал, у вас больше не работает и поэтому физически не может помочь новому сотруднику в нем разобраться. Итак, у вас два выхода. Первый: пасть под давлением минималиста, сделав его самым счастливым человеком, но загубив последний шанс сдать проект в срок. Второй: направить его на путь изучения существующей архитектуры и дать ему впоследствии возможность внести в нее весомый вклад. Поиграйте с его пристрастием к четкой архитектуре — попросите документировать существующий код с тем, чтобы в будущем, если позволит время, он смог бы переписать некоторые объекты, сделать их более удобными. Если он профессионал, не сомневайтесь — он обязательно разберется в том, что написал его предшественник. Не стесняйтесь использовать в своих целях дух соревновательности. С точки зрения минималиста все, что написал не он, — полная туфта. На самом деле вполне возможно, что он боится, будто не сможет разобраться в существующем коде, и просто не хочет в этом признаться. Всегда ищите скрытые мотивы, которыми руководствуются все без исключения представители рода человеческого. Поймите: программисты, не готовые признать, что их компетенции не хватит для решения поставленной задачи, очень любят выискивать своему бездействию высокоинтеллектуальные оправдания.

Как поступить с архитектором, который уверен, что созданные им объектные решения превосходят все созданное ранее, а вы, тем не менее, усматриваете в них некие слабые стороны? Не надо ему ничего говорить про врожденные недостатки решений — ничего не добьетесь, зато наживете себе врага. Лучше попросите его объяснить предполагаемый механизм функционирования всех динамичных элементов и построить несколько прототипов, или тестовых программ, которые смогли бы наглядно продемонстрировать действие заложенных в решении функций. Если он создаст эти прототипы и никаких проблем не возникнет, значит, возможно, вы были неправы, и изъянов в найденном решении нет. Если архитектура кажется вам слишком громоздкой и монолитной, попросите разбить ее на компоненты. Если окажется, что они прекрасно друг с другом работают, значит, архитектор вполне адекватно представляет себе, что он хочет. Если объекты излишне взаимосвязаны и взаимозависимы, это свидетельствует о пристрастии архитектора к усложнению, которое способно существенно удорожить сопровождение продукта. Что отличает высокопрофессионального архитектора? То, что он способен создать конструкцию, которую сможет обслуживать и расширять любой его последователь. Такая конструкция не рассыплется при первой же модернизации кода. На код при работе с архитектором нужно смотреть его глазами — даже если он слеп на один глаз и не видит другим.

Есть твердое правило: прежде чем пытаться утвердить то или иное решение, используя свое положение руководителя, обязательно выслушайте человека и попробуйте его понять.

Какие еще рекомендации я мог бы дать по поводу такого рода ситуаций межличностного общения? Есть твердое правило: прежде чем пытаться утвердить то или иное решение, используя свое положение руководителя, обязательно выслушайте человека и попробуйте его понять. В том, что касается конфронтации, программисты ничем не отличаются от остальных людей — они хотят, чтобы их выслушали. Как пишет в своей книге «The 7 Habits of Highly Effective People» Стивен

Кави (Stephen Covey), «сначала старайтесь понять... и только после этого — быть понятым». Поиск консенсуса при принятии технических решений есть не что иное, как вид искусства, основывающийся на готовности выслушивать чужие идеи. Для того чтобы выстроить такую основу для взаимодействия с сотрудниками, требуется терпение, и проявлять его необходимо — хотя иногда нам кажется, что времени нет даже на конструирование, а насчет методов все вроде бы согласны. Вы можете так считать, но это не отменяет постановку задачи по достижению консенсуса[1]. О том, как достичь консенсуса, мы еще поговорим в главе 5, которая посвящена проведению проектных совещаний. Возможно, вы удивитесь, когда узнаете, что консенсус нельзя строить на основе компромисса.

И еще один пример, иллюстрирующий принцип «услышь, прежде чем судить». Некоторые языки программирования — и, в частности, Visual Basic (VB) — не предусматривают полноценных конструкторов объектов. Недавно я столкнулся с тем, как один художник с помощью события инициализации класса VB пытался организовать обращения к набору объекта со стороны (родительского) класса-потребителя[2]. Если объект VB не удается конкретизировать, перехватить ошибку становится очень трудно. Когда я спросил этого деятеля, почему для обработки подверженной ошибкам операции он выбрал упомянутое событие, в ответ он сообщил, что его решение изящно, понятно и не требует от вызывающего объекта подготовки обращения к интерфейсу. Я, естественно, посчитал, что надежная обработка ошибок значительно важнее любых попыток вылизать текст. Но промолчал. Ознакомившись с его аргументацией, я объяснил ему, с какими трудностями может столкнуться такой объект, и подкрепил свои соображения наглядным примером в коде. Если бы я сразу сказал что-нибудь вроде «это не есть правильно — разберись!», то не смог бы образумить его своим примером, и он так бы не понял, чего от него хотят. Повторюсь: если дать человеку возможность высказать его точку зрения, он в ответ проявит понимание к вашей позиции.

Как вы адаптируетесь

Из этой главы вы почерпнули множество новых идей. Возможно, вам немного не по себе от масштаба изменений, которые приходятся на долю руководителя на пути самосовершенствования. Не беспокойтесь. В конце концов, на 99 % люди до сих пор генетически идентичны обезьянам, а оставшийся процент сформировался далеко не сразу. Последующие главы, в которых раскрываются другие аспекты руководства и менеджмента, помогут вам адаптироваться, преодолеть трудности и достичь успеха.

Теперь повторим пройденный материал — благо основные принципы руководства должны обосноваться в вашей голове надолго.

- *Вы должны уметь адаптироваться.* Нарабатывая навыки руководства, человек должен приспосабливаться к новому для него социальному контексту. Вы —

---

[1] Существует мнение, согласно которому полное и всеобщее согласие приводит лишь к тому, что в случае неудачи не остается никого, кого можно было бы в ней обвинить. Может, это и так, но, будучи руководителями, мы не должны увлекаться поиском виновных — значительно полезнее посвятить свое время решению проблем.

[2] Событие инициализации в VB не принимает и не возвращает никаких параметров.

начальник, а потому ваши взаимоотношения с подчиненными должны серьезно измениться.

- *Самосовершенствование важнее, чем имидж.* Лидерство есть внутреннее качество человека, оно ни в коем случае не обусловливается внешними признаками. Оставаясь программистом, вы как руководитель должны работать над решением сложных проблем, предусматривающих анализ поведения подчиненных (и, конечно же, собственного поведения).
- *Изучайте своих сотрудников.* Станьте исследователем культуры и личностей программистов. Разберитесь, почему ваши подчиненные пишут код именно так, как пишут; подумайте, как использовать их положительные качества и улучшить положение вещей в неблагополучных с точки зрения производительности областях. Пасти котов — значит заставлять их двигаться в одном направлении. Это основная задача руководителя.
- *Вознаграждайте сотрудников согласно их заслугам.* В вопросах устного и денежного поощрения действуйте по ситуации. Принимая во внимание все финансовые ограничения, старайтесь быть одновременно справедливым и бережливым. Не забывайте, что хвалить подчиненных лучше публично, а ругать индивидуально.
- *Думайте.* Мобилизуйте ваши знания о людях и на их основе старайтесь находить консенсус. Прежде чем судить, выслушивайте аргументацию собеседника. Воспитывайте свой мозг — поскольку вы теперь руководитель, размышления должны стать вашей второй натурой. Готовые варианты решения личностных проблем не способны заменить ваших стараний, направленных на устранение трудностей и развитие возможностей, характерных именно для *вашей* компании.

Что дальше

В этой главе мы анализировали ваши кадры; в следующей будем изучать вас. Я задам несколько довольно трудных вопросов, призванных выявить ваше отношение к роли руководителя в высшей степени важного процесса конструирования программного обеспечения в контексте текущей конъюнктуры. Чтобы стать хорошим руководителем, вы прежде всего должны научиться управлять собой.

Глава 2
Как руководить собой

Создание в срок качественного программного обеспечения — ваша цель, а управление процессом — ваша работа, так что же вы делаете в свободное время? Вероятно, пишете код. Если вы доросли до начальника из программистов, вам стоит продолжить программировать, в противном случае ваше увлечение этим ремеслом будет ослабевать, и ваши навыки постепенно «сойдут на нет». Другой возможной причиной для того, чтобы искать и находить время для кодирования, является удовольствие, которое оно доставляет. Если возглавлять команду программистов — для вас дело новое и вы пока еще адаптируетесь к этой роли, то кодирование доставит вам удовольствие, поскольку именно это вы хорошо знаете и умеете делать продуктивно. Я полагаю, что продолжать писать код абсолютно необходимо, поскольку это занятие связывает вас с прошлой жизнью, с вашими корнями. Корни очень важны, ведь они определяют ваше отношение к своему ремеслу, и как руководителю вам придется пустить в грунт своей жизни несколько новых корней.

Исследование самого себя даст начало этим корням, так что занимайтесь самоанализом во время ваших ночных бдений над кодом. В предыдущей главе мы рассмотрели принципы взаимодействия с подчиненными, теперь пришло время взглянуть на того, кто ими руководит, — на вас. В этой главе рассматриваются вопросы руководства собой. Каким руководителем я являюсь? Под кого надо подстраиваться, под начальников или под подчиненных? А может, одновременно под тех и других? Ответы на эти вопросы чрезвычайно важны для вашего роста и успеха как руководителя.

Взгляд в зеркало

Исследование вашей объективности может оказаться трудной задачей, поскольку в этом процессе существенную роль играет субъективность. Для того чтобы

облегчить самопроверку, взгляните на следующий перечень вопросов и обдумайте их по ходу чтения этой главы. В ее конце я предложу свои ответы на них. Эти ответы я считаю естественными для управленца, и они помогут вам стать лучшим руководителем, чем вы есть. Как теннисист, вы не можете сделать хороший замах, не сжав рукоять ракетки. Управление — это ракетка, так что предложенные вопросы требуют ваших ответов. Итак:

- Считаете ли вы, что предельные сроки завершения проекта — это рекламный прием, на самом деле не имеющий большого значения?
- Позволяете ли вы программистам самим принимать основные архитектурные решения, если вы слишком утомлены или заняты?
- Надеетесь ли вы, что ваши опытные программисты и без вас все сделают правильно, поскольку у вас просто нет времени на то, чтобы нянчиться с ними?
- Полагаете ли вы, что при приближении срока сдачи проекта все проблемы в конце концов решатся[1]?
- Считаете ли вы, что пользователи никогда не сделают ничего, что привело бы к программному сбою, просто потому, что у них не хватит ума для этого?
- Предпочитаете ли вы при управлении коллективом искать консенсус, даже если это требует массы времени и терпения?
- Считаете ли вы электронную почту эффективным средством общения при работе над проектом?
- Согласны ли вы проговорить по телефону несколько часов кряду, только чтобы убедиться, что все идет как надо?
- Считаете ли вы, что с помощью комитетов и комиссий невозможно выработать адекватные бизнес-требования?
- Считаете ли вы себя самым умным программистом в компании?
- Чувствуете ли вы ревность и страх, когда смотрите на действительно хороший код, который написан не вами?
- Делаете ли вы все возможное, чтобы успеть закончить проект в срок[2]?

Ответы, которые вы даете на эти вопросы, могут меняться в зависимости от обстоятельств[3]. Тем не менее некоторые из ответов правильны вне зависимости от текущей ситуации на административном поприще. Надеюсь, что последующие разделы подготовят вас к тому, чтобы отвечать не задумываясь. Возможно, с некоторыми истинами вам будет трудно согласиться, но я полагаю, что, познав самого себя, вы сможете принять их и действовать в соответствии с ними.

---

[1] Это то же самое, что предпочесть синюю пилюлю красной в «Матрице», культовом фильме многих программистов.

[2] Южанин бы сказал: «Я пойду за этим в пасть дьявола». Это значит, что вы ни перед чем не остановитесь ради того, чтобы достичь своих целей. Природа этой аллегории происходит от изображения ада в виде ужасной разинутой пасти, готовой поглотить вас, — очень напоминает срок сдачи проекта, к которому невозможно успеть. Вы проявляете твердость и с мечом в руках рубитесь у врат ада, преодолевая силу, стремящуюся вас уничтожить.

[3] Помните «Звездные войны»? Что Оби-Ван, говоря об отце Люка, сказал о природе правды и точки зрения? (Когда умер Йода.)

Рай, ад, чистилище и ваше место во вселенной

Наиболее существенное усовершенствование, которое вы можете сделать, руководя разработкой программного обеспечения, — усовершенствовать руководителя.

Наиболее существенное усовершенствование, которое вы можете сделать, руководя разработкой программного обеспечения, — усовершенствовать руководителя.

Уильям Блейк (William Blake) писал: «Тот, кто желает, но не действует, распространяет чуму»[1]. Если вы не исправите слабые места в вашем стиле руководства, вы распространите чуму среди своих программистов. Продолжая наш поэтический экскурс (поэзия — близкая родственница программирования), отметим следующие строки:

...в сердце взгляни,

Растет в нем святое древо,

Листья дрожат, как огни,

Питает их радости чрево[2].

Вы должны ухаживать за этим «деревом». Как Вергилий, ведший Одиссея сквозь чистилище[3], я здесь для того, чтобы провести вас к вашему новому месту во вселенной программного обеспечения.

Осознание вашего места в происходящих вокруг вас событиях поможет вам стать лучшим руководителем, чем вы были прежде. Давайте оглядимся в этой новой и прекрасной вселенной.

Ваша работа в корне меняется

Ну что ж, вы не в раю, это уж точно. Наверняка вы осознали это в первый же месяц, распределяя задачи в коллективе. Хотя вы можете подумать, что в раю живет ваш начальник, смею вас уверить, что и это не так. Вы видите, как он планирует текущие задачи и проводит планерки на уровне компании, касающиеся отдаленного будущего, но вы никогда не видели, чтобы он делал что-то, имеющее какую-либо ценность само по себе. Ваше впечатление ошибочно и показывает, насколько вы нуждаетесь в изменении точки зрения на выполняемую работу. Вскоре мы подробнее поговорим об этом, в особенности в главе 9, где

---

[1] Эту и многие другие прелестные вещи, в равной степени поучительные и гнетущие, можно найти в поэмах Блейка «Marriage of Heaven and Hell». Упомянутая выше цитата взята из «Proverbs of Hell» — William Blake, The Complete Poetry and Prose of William Blake, ed. David Erdman (Derkeley, CA: University of California Press, 1982).

[2] Уильям Батлер Йейтс (William Butler Yeats), избранные поэмы (New-york: Collier Books, 1986). Перевод В. А. Савина (http://zhurnal.lib.ru/s/sawin_w_a/rtfrtf.shtml).

[3] Вы можете освежить ваше классическое образование, прочтя современный перевод Чистилища Данте. Когда вы окажетесь в роли руководителя, вам может показаться, что вы находитесь где-то между раем и адом, однако в действительности это не так — это наше видение обычной жизни.

наше внимание будет сфокусировано исключительно на отношениях с начальством.

Вы уже поняли, что кодирование больше не ваш хлеб. Иногда вы относитесь к этому как к «программистскому аду», поскольку получаемые вами задания должны быть выполнены к нереальным срокам, и вам не с кем посоветоваться, кроме себя. Как программист вы привыкли к жаре, получали удовольствие от общения с вашими собратьями-программистами и считали программирование основным делом своей жизни.

Теперь у вас появилось другое место — место начальника программистов. Вы где-то посередине между раем и адом и поэтому наслаждаетесь преимуществами обоих. В мифологии такое промежуточное место между раем и адом называется чистилищем. Именно в нем устраняются последние преграды на пути в рай. В чистилище грешник также получает свою долю мук, но не таких ужасных, как в аду. На первых порах вы можете не почувствовать, что в этом новом месте страдаете меньше, но когда вы однажды к нему привыкнете, оно вам покажется совсем неплохим. На самом деле я не знаю лучшего места: вы по-прежнему можете писать код, но у вас также есть возможность помочь другим делать ту же работу.

Вам нужно заново учиться оценивать свои успехи, увлечения, амбиции

Геометрические аспекты рая, чистилища и ада вполне применимы к вашему месту в иерархии управления компанией. Представьте себе, что с каждой ступенькой вверх по административной лестнице растет и высота, с которой вам, возможно, придется падать. Если ваша лестница правая (прислонена к правой стороне стенки), мужайтесь: в конце концов, вы знаете, что ваше дело правое. Кстати говоря, а куда вы движетесь как программист-начальник? Будем надеяться, по направлению к успеху — как личному (персональному), так и общему (корпоративному). Хотя успех и определяется по-разному, одно из определений я нахожу наиболее полезным и практичным: это способность радоваться своему труду и не терять своего увлечения. Может показаться, что при такой оценке увлечение ставится слишком высоко, однако оно может зажечь огонь у вас внутри и помочь вам покорить столь непредсказуемый мир разработки программного обеспечения. Увлечение — это топливо, которое может раскрутить «мотор» вашего руководства. Требуйте от своих программистов во всем добиваться безупречности, чего бы это ни стоило. Если они считают, что изящество и безупречность — это те две вещи, которые затягивают время кодирования, напомните им, что элегантность — это достижимое совершенство. Увлечение побудит их добиться этих целей. Лелейте свое увлечение — ведь для вас это единственный способ расти, приспосабливаться, превозмогать, достигать цели. Это часть обязанностей по уходу за деревом, на котором распускаются цветы успеха.

Лелейте свое увлечение — ведь для вас это единственный способ расти, приспосабливаться, превозмогать и достигать цели. Это часть обязанностей по уходу за деревом, на котором распускаются цветы успеха.

Учитесь обживать свое место, а не рваться вперед. Амбиции могут быть чрезвычайно разрушительной силой, если они отрывают вас от реальности и решаемых задач. Быть амбициозным — значит преуспевать в вашей новой роли руководителя и отчасти программиста. Если в будущем вам выпадет продвинуться по служебной лестнице, вы должны быть уверены, что продвижение по службе — совсем не то, к чему вы активно стремитесь, а просто лучший способ реализовать свои амбиции. Вы сможете делать эту амбициозную работу, если у вас сердце программиста и при этом вы сумели развить в себе мышление руководителя.

Вы сможете делать эту амбициозную работу, если у вас сердце программиста и при этом вы сумели развить в себе мышление руководителя.

Естественный отбор и время

Из наблюдений за миром живой природы мы знаем, что для естественного отбора, искореняющего недостатки и повышающего выживаемость, время — один из необходимых ингредиентов. В мире программного обеспечения у вас нет такой роскоши, как тысячелетия. Потребителям программного обеспечения может показаться, что на создание новой версии уходит бездна времени, но они ведь просто не знакомы с процессом. Вы разбираетесь в процессе и должны учитывать время в повседневной деятельности. В революционной книге, посвященной вопросам управления разработкой программного обеспечения, пустая трата времени персоналом упоминается как величайший грех управленца[1]. А как вы тратите ваше собственное время руководителя?

Время — это либо союзник, либо враг. Взвесьте, что сказал Френсис Бэкон (Francis Bacon) — один из отцов-основателей современной науки:

> «Тот, кто не ищет новых лекарств, должен ожидать появления новых бед, ведь время — величайший новатор»[2].

Немного помогите процессу естественного отбора: продумывайте административные вопросы, которые грозят обернуться потерей времени, вносите изменения и работайте на опережение, иначе вы рискуете оказаться в их власти. В этом разделе я опишу несколько проблем подобного рода. Я не стану останавливаться на деталях управления, они будут обсуждаться в следующих главах. На данный момент я хочу только, чтобы вы поняли: сознательно или нет, но некоторые методики вы уже применяете. Время, растрачиваемое впустую, должно быть вашей постоянной головной болью.

---

[1] Tom DeMarco and Timothy Lister, Peopleware: Productive Projects and Teams, Second Edition (New York: Dorset House Publishing, 1999).

[2] См. эссе Бэкона «Of Innovations» (1625). Еще одна занятная цитата: «Время — это огонь, в котором мы горим». Это, возможно, взято из произведения какого-нибудь великого классического писателя, но черт меня возьми, если я смогу выяснить, кого именно, так что нам пришлось ограничиться Бэконом.

Избегайте ненужных, неэффективных совещаний

Как менеджер и руководитель вы будете участвовать в куда большем количестве встреч, чем в бытность свою программистом. Глава 5 целиком посвящена этому предмету. Давайте обсудим сейчас, какие из этих встреч действительно необходимы, и постараемся сделать их эффективными.

Для общения вашего коллектива вполне достаточно одной встречи в неделю. Остерегайтесь тратить чересчур много времени на разговоры и мало на решения и действия. Не устраивайте встречу только ради того, чтобы получить одобрение своих решений. Поощряйте дискуссию, но ищите решения. И помните, что дьявол — в деталях, а цель любой встречи — это изгнание этих дьяволов. Если вы контролируете ход встречи, вы контролируете время. Установите предел для любой встречи: 45 минут общения обычно более чем достаточно. Зачастую могут быть нужны и 8-часовые обсуждения проекта, однако всегда имейте подробную повестку дня и следуйте ей, если вы собираетесь выдержать такую длительную мозговую атаку.

Не устраивайте встречу только ради того, чтобы получить одобрение ваших решений. Поощряйте дискуссию, но ищите решения.

Не планируйте слишком мало или слишком много

Не верьте теории, утверждающей, что чистый стол — признак скудоумия. При взгляде на рисунки Эйнштейна может показаться, что на столе у него царил ужасный беспорядок, но я ручаюсь, что в его голове — нет. Вам необходимо в достаточной мере организовать процесс, но не тратить все время на планирование. Достичь равновесия трудно, но необходимо. Тема организации, необходимой для успеха, затронута глубже в главе 4. Не путайте вопросы организации работы с вопросами ее выполнения. Организация — это начальная стадия плана. Разрабатывайте план. Вспомните, что сказал Спок (Spock): «Логика — начало мудрости»[1]. Аналогично, планирование — начало выполнения.

Бессмысленно ожидать чего-либо при отсутствии контроля

Вы роздали задания, вернулись к вашим обязанностям и надеетесь, что все заняты выполнением заданий. Однако после ваших подробнейших объяснений Джо спокойно отправляется бороздить Интернет, даже не попытавшись заставить программный интерфейс приложения работать правильно. Если вы думаете, что промежуточные этапы не являются неотъемлемой частью поставленной вами задачи, то вы понятия не имеете о том, как работает голова программиста, даже несмотря

[1] Помните как в «Звездном пути» Спок напился с Валарисом после их беседы об изгнании людей из рая? Гм, возможно, рай — это просто работа программиста в сравнении с работой руководителя, — вам решать.

на то, что вы — один из них. Это вполне в человеческой природе: если до срока сдачи остается месяц, времени на то, чтобы сделать работу, вполне достаточно. Время можно считать потерянным, если нет очевидного продвижения. Работа над частью нового продукта — это поступательный итерационный процесс, не подчиняющийся булевой логике и предполагающий много возможностей для проверки. Предположим, вы сообщили отделу тестирования, что дата завершения кода — XX-е января. Лучше, если эта дата не будет рабочей датой окончания работы над кодом! Вы не сможете сделать работу X, если сначала не будет выполнена часть X – Y, где Y — еженедельно определяемая часть работ. (Вы можете сказать, что «сделано за день»?) Вы наверняка слышали, что цена свободы — это постоянная бдительность, ценой же вовремя сделанного программного обеспечения является неизменное усердие.

Вы наверняка слышали, что цена свободы — это постоянная бдительность, ценой же вовремя сделанного программного обеспечения является неизменное усердие.

Проектируйте архитектуру, прежде чем выбирать технологию

Технология волшебной пули или золотого молотка (как бы вы ее ни назвали) не может решить бизнес-проблем, это делают люди. Уверен, вы применяете технологию для реализации решений, но вы тратите время попусту, если думаете, что покупка последнего дополнения к среде разработки даст скачок производительности. Следующая версия языка программирования также не решит ваших проблем. Конкурирующие производители средств разработки программного обеспечения обещают многое. Наша отрасль разделена надвое: Microsoft и все остальные. Я, конечно, понимаю, что это слишком упрощенное разделение, но оно послужит иллюстрацией моего утверждения: продукты Microsoft могут оказаться не оптимальными для решения ваших конкретных задач, так как Microsoft слишком большая и многоплановая корпорация. Не важно, много или мало у Microsoft возможностей влиять на всю отрасль и каковы эти возможности, — технология Microsoft построена на базе заранее определенного архитектурного плана. Мир Java, олицетворяемый Sun, слегка отличается, и хотя он более фрагментирован по сравнению с Microsoft, Sun также создает свои продукты на основе конкретной архитектурной схемы. Вы можете использовать Enterprise JavaBeans или .NET сколько душе угодно, но в любом случае вам придется принять внутренние архитектурные ограничения. Я призываю определить ваши архитектурные задачи и планы до того, как вы выберете технологию реализации. Вам придется все переделать, если с новым инструментом дело «не выгорит». Вы уже много раз слышали: если у вас не хватает времени даже на то, чтобы сделать все правильно, где же вы его найдете на то, чтобы все переделать?

Баланс между чистотой и практичностью

Поиск равновесия между чистотой и практичностью — трудное дело для человека, влюбленного в программирование. Концепция «хорошего программного обеспечения нужно в меру», возможно, впервые открытая Microsoft, имеет свои достоинства. Вы можете тратить очень много времени, добиваясь чистоты кода в ущерб практичности. Что действительно нужно, так это удобство эксплуатации программного обеспечения, и практичность больше соответствует этой цели, чем чистота. Конечно, чистота может быть полезной, но какой ценой? Я не говорю о создании неряшливого или непродуманного кода, я имею в виду программное обеспечение, которое может дополняться и расширяться не только его создателем, но и другими программистами. Проблема чистоты кода состоит в том, что она подобна красоте — ее воспринимают глазами. Вашим глазам постоянно нужно носить очки практичности.

Не выполняйте задания, а распределяйте их

Классическая управленческая ловушка для начальника, вышедшего из программистов, касается распределения заданий. Вы понимаете, каким образом реализовать определенное решение, а обучение членов команды, которые вовсе не обязательно видят это решение, требует времени. Здесь применима старая китайская поговорка: «Дайте человеку рыбу, и он будет сыт один день; научите его ловить рыбу, и он будет сыт всю жизнь». Если бы все стали столь же сообразительны, как вы, было бы вам проще работать? Возможно. Потратьте время на обучение сейчас, и вы сэкономите его потом, поскольку ваши сотрудники научатся решать проблемы без вашей помощи. Тогда вы сможете эффективно распределять задания, а не давать объяснения.

Документируйте то, что вы делаете или планируете делать

Трудным заданием для тех, кто любит программировать или управлять по наитию, является документирование. При необходимости можно «опередить события» и сделать экранный прототип, но экранные снимки делаются только ради конкретизации проектной документации. Не отдавайте прототип напрямую программисту, который должен завершать проект. Может показаться, что такой прототип сэкономит время, но он может просто повести программиста неверным путем и спровоцировать фальстарт. Не думайте, что для написания кода бизнес-требований достаточно. Бизнес-требования — только начальный этап разработки документа, позволяющий обрисовать архитектуру, а дальше вы можете говорить о реализации решения в объектах кода. У вас всегда будет искушение, когда поджимает время (а разве бывает иначе?), слепить все «по-быстрому», однако надо быть настоящим счастливчиком, чтобы, не имея четкого плана, создать программный продукт, который можно будет в дальнейшем дополнять и обновлять.

Вы можете время от времени ощущать себя во власти блок-схем и диаграмм, но это лучше, чем код, хранящийся в корпоративной базе данных без какого-либо указания на то, как он работает и какую проблему он призван решать. Документирование радикально отличается от программирования, но это необходимый первый шаг на пути превращения идей в продукты. Следующим шагом может быть прототип, но смотрите на прототип как на продолжение документа, а не как на начало проекта. Прототипы служат для обоснования готовой концепции, в то время как программные продукты представляют собой реализацию готового проекта.

Прототипы служат для обоснования готовой концепции, в то время как программные продукты представляют собой реализацию готового проекта.

Оценка вашей производительности

Как я упомянул в этой главе ранее, в момент, когда вы впервые превращаетесь из программиста с полной занятостью в начальника с полной занятостью и одновременно программиста с частичной занятостью, административная деятельность кажется не столь продуктивной, как кодирование. Учитесь различать просто трату нервной энергии и настоящее творчество. Что я имею в виду? Если вы привыкли кодировать часами, то вы поймете, что такое нервная энергия. Это то состояние вашего ума, когда вы прикончили уже три чашки кофе[1], вам еще писать и писать, но вы уже чувствуете приближение результата. Настоящее творчество — это ощущение, когда вы видите и конечный результат, и все ступени его достижения, но при этом знаете, что должное качество будет достигнуто далеко не так быстро. Я знаю, что все сказанное трудно переварить, но вам действительно нужно как следует обдумать эту концепцию.

Как программист вы привыкли измерять свою производительность числом работоспособных объектов, созданных вами в течение дня. Подобный метод оценки может стать причиной вашего провала как руководителя. Вы будете разочарованы и не удовлетворены тем, как вам приходится проводить время, пока не примете новый образ мышления. Как начальнику вам следует оценивать свою производительность объемом работы, которую выполняет ваш коллектив. Вы не можете оценивать этот объем каждый день, и это может стать серьезной проблемой, если вам требуются постоянные подтверждения того, что вы работаете хорошо. Как ежедневно решать эту проблему? Заходите к своим программистам каждый день или звоните им и проверяйте, как идут дела. Когда они привыкнут ждать этих ежедневных встреч, произойдут две вещи: они никогда не будут знать, когда вы появитесь, и, таким образом, постараются быть в «постоянной готовности», а вы сможете оценить их ежедневный прогресс и соответствующим образом подстроить под него свой график работы. При этом обе стороны прекрасно

[1] Подставьте сюда любую любимую вами жидкость, содержащую кофеин. Я предпочитаю кофе програм мистской крепости, светонепроницаемый и способный почти стоять на столе без всякой чашки.

понимают, что происходит, однако приятно притворяться, что никто ничего не замечает.

Как начальнику вам следует оценивать свою производительность объемом работы, которую выполняет ваш коллектив.

В своей крайне проницательной книге «The End of Patience» Дэвид Шенк (David Shenk) пишет:

> «Что касается гипертекста, окончания излишни, поскольку никто никогда их не достигнет. Чтение открывает дорогу к катанию на волнах информационного океана, извилистому, бесконечному странствию через лабиринт тем. Странник создает собственную повесть, выбирая наиболее привлекательную ссылку, которая всегда доступна. Как техника поиска — это роскошно. Однако как способ мышления это имеет серьезные пороки»[1].

По аналогии с приведенной цитатой, не оценивайте вашу производительность тем, насколько быстро вы можете перепрыгнуть от одного проекта (или сотрудника) к другому. Не теряйте из виду масштабные цели и оценивайте небольшие шаги, необходимые для успешного достижения цели. Не занимайтесь самообманом — пусть реальное положение дел диктуется фактами, а не вашим оптимистичным воображением. Ключевые слова из предшествующей цитаты — «привлекательный» и «мышление». Не соблазняйтесь непосредственным удовольствием от кодирования; лучше поразмыслите, как помочь остальным получать это удовольствие под вашим руководством.

Как я уже упоминал, управление программистами и разработка программного обеспечения требуют серьезных размышлений. Добавьте к этому перечню настойчивость, качество, которое отнюдь не всегда присуще душе программиста, но которое в случае крайней необходимости может быть привито ей программистом-начальником. Примените часть этой настойчивости к самому себе — и вы обна- ружите, что она нужна как программисту, так и его начальнику. Системность — внутренняя дисциплина, привнесенная вами в работу, — помогает быть настойчивым.

Если вы новичок в деле управления, не ожидайте, что почувствуете себя комфортно ранее, чем через 6 месяцев. Пусть этот дискомфорт напоминает вам о необходимости многому научиться, а также о том, что изучение новых способов быть полезным и *чувствовать себя* полезным требует времени, но за это воздастся в будущем.

Контролируйте свои слабости

Вы великий программист, правда? Я уверен в том, что так оно и есть, а также в том, что вы все же еще не совершенны. Остерегайтесь, что ваши слабости как программиста заставят вас смотреть сквозь пальцы на тех в вашем коллективе,

[1] David Shenk, The End of Patience: Cautionary Notes on the Information Revolution (Bloomington, IN: Indiana University Press).

кто похож на вас. Если вы сами не любили документировать и проектировать, прежде чем начать кодировать, тогда вы, вероятно, можете позволить другим этого не делать.

Если вы не любите документировать и проектировать, прежде чем начать кодировать, тогда вы, вероятно, можете позволить другим этого не делать.

Такие вещи вряд ли могут быть полезными. Если вы минималист, то в случае когда дело доходит до обработки ошибок (которых в идеале не бывает), вы можете не заметить отсутствия процедур обработки ошибок в коде Салли, считающей все настолько очевидным, что обработка ошибок — это просто трата места.

Я могу продолжить перечень слабостей программистов, но оставлю это вам. Тем не менее вам требуется осознать необходимость постоянно быть внимательным к своим недостаткам как программиста, поскольку нельзя прощать другим то, что вы могли бы простить себе. Это может выглядеть противоречиво, и так оно на самом деле и есть, так что продолжайте работу над исправлением своих недостатков как программиста, чтобы не заразить ваших подчиненных теми же слабостями.

Чтобы вам было легче преодолевать свои слабости, представьте себе, что многие люди прошли тот же путь, что и вы. Некоторые из них проложили на этом пути вехи для вас — это книги и иногда URL-адреса. Другими словами, для того чтобы заполнить пробелы в знаниях и/или опыте, рекомендую побольше читать. Вам нужно читать по многим причинам. Вот некоторые из них.

- *Читайте, чтобы оставаться «на уровне».* Сюда можно отнести чтение отраслевых журналов, книг, посвященных вашему основному языку программирования, и журналов, касающихся методов, применяемых в вашей работе. Это вы должны делать почти каждый день, просто чтобы не потерять необходимые навыки.
- *Читайте, чтобы быть в курсе.* Такое чтение расширит ваши познания, причем некоторые из источников, используемых для сохранения вашей квалификации (чтобы оставаться «на уровне»), применимы и в этом случае. Здесь вам надо сфокусироваться на том, что, возможно, не требуется прямо сейчас, но может понадобиться в ближайшем будущем. Интернет-рассылки, форумы и прочие чудеса информационной эры могут быть необычайно полезными. Не забывайте, однако, что эра информации — это совсем не то же самое, что эра знаний.
- *Читайте, чтобы углубить свои знания.* Подобное чтение может быть сложным, но вы должны находить для него время, чтобы достичь более полных знаний по технологии и методикам, с которыми сталкиваетесь в своей повседневной работе. Здесь вам придется внимательно изучить книжные шкафы в вашей местной библиотеке, книжном магазине или Интернете в поисках книг, которые действительно могут помочь. Для того чтобы углублять свои знания, в выборе книг избегайте характерного для поваренной книги подхода к изложению. Вам нужно нечто большее, чем просто набор рецептов; вам нужно «откусывать больше того, что вы, как вам кажется, можете прожевать».

- *Читайте, чтобы стать мудрее*. Умные люди встречаются не только в нашей области. Другие научные дисциплины, а также художественная литература могут научить вас подходить к любому вопросу с разных сторон. Очень часто в нашей профессии мы ограничиваемся вертикальным мышлением. Вертикальное мышление подразумевает поступательное движение от одного логического умозаключения к следующему и отбрасывание того, что не укладывается в рамки заранее имеющихся у нас представлений о наиболее подходящем. Разностороннее мышление — это то, что приносит оригинальные идеи и зачастую является признаком гениальности.

Хочу специально сообщить приверженцам технологии: бизнесмены — тоже умные люди. Под бизнесменами я подразумеваю тех, кто многие годы участвует в строительстве крупных корпораций и правительственных органов. Должен признать, что мне далеко не сразу удалось научиться уважать навыки, необходимые для ведения бизнеса. Я был интеллектуальным снобом, считавшим, что если кто-то не понимает всех деталей технологии, то он по определению не может принадлежать к когорте ярчайших и лучших. Это было проявлением слабости. Вы можете многому научиться у руководителей в различных областях, не только связанных с технологией. Расширьте ваши интеллектуальные поиски, включив в них области деятельности, в большей степени связанные с применением не технологии, а деловой хватки. Концентрировать усилия людей на решении бизнес-проблем — ваша основная работа как начальника, а технология — это только одно из возможных решений. Вам нужно ознакомиться с другими путями решения проблемы, так что читайте книги, в том числе и не относящиеся к области технологии.

Вот пример того, что я имел в виду, говоря о деловом подходе. Джек Уэлч (Jack Welch), бывший долгое время главой General Electric, пишет:

> «Я полагаю, что бизнес во многом похож на ресторан мирового класса. Если вы заглянете в двери кухни, пища никогда не будет выглядеть так же хорошо, как на вашем столе, куда она попадает великолепно украшенной, в красивой посуде. Бизнес беспорядочен и хаотичен. Я надеюсь, что вы сможете найти на нашей кухне что-нибудь, что могло бы быть полезным в достижении вашей мечты»[1].

Не правда ли, сказано будто про бизнес в области программного обеспечения? Уэлчу есть много чего сказать, причем вполне уместного и для руководителя программистов, поскольку он пишет о руководстве в течение 20 лет одной из крупнейших и доходных компаний. Учитесь у тех, кто проделал путь наверх до вас. Идите по их следам.

Многочисленные ссылки на литературу, встречающиеся на страницах этой книги, призваны помочь вам лучше ориентироваться. Выберите полезные для вас издания и прочитайте их. Книги нужны не только для того, чтобы украшать полки или произвести впечатление на ваших друзей. Хорошие книги — это жизненные откровения. Научитесь определять, кто из авторов обращается непосредственно к вам, и слушайте то, что он пытается до вас донести.

---

[1] Jack Welch, Straight from the Gut (New-York: Warner Business Books, 2001), p. xv.

Ответы

Вопросы, с которых начиналась эта глава, были задуманы как трамплин для самоанализа. Теперь, когда вы, оттолкнувшись двумя ногами, прыгнули в глубину своей души, давайте познакомимся с ответами на эти вопросы.

- Считаете ли вы, что предельные сроки завершения проекта — это рекламный прием, на самом деле не имеющий большого значения?

  На самом деле сроки всегда имеют значение. Ваша компания может распространять программное обеспечение с целью продажи предоставляемых им услуг, и информация о товаре должна содержать описание его свойств. Даже если пользователи не задействуют все возможности вашего продукта, они должны быть представлены в рекламном буклете. Некоторым состоятельным клиентам может потребоваться лишь одна деталь, которая не нужна остальным, и ее поддержка программным обеспечением может стать причиной ощутимой выгоды. Маркетинг может оказывать опасное влияние на разработку программного обеспечения, однако он необходим. Даже если вы не хотите, чтобы весь цикл разработки определялся отделом продаж, все равно для вас он будет одной из движущих сил в определении даты окончания работы над проектом. Постарайтесь узнать продавцов поближе, стать их другом. Заслужите их доверие пунктуальностью и узнайте от них положение вашего продукта на рынке. (См. далее врезку «Кошачьи разборки», непосредственно связанную с этим вопросом и ответом на него.)

- Позволяете ли вы программистам самим принимать основные архитектурные решения, если вы слишком утомлены или заняты?

  Вам, скорее всего, придется перепоручать принятие некоторого количества непростых решений, особенно в первые дни вашей карьеры руководителя. Это может сработать, особенно если вы руководите хорошими людьми, однако помните, что возможность перепоручить принятие решений есть у вас не потому, что вы устали, а потому, что лучшие решения могут рождаться не только в вашей голове. Отказ от принятия решения — тоже решение. Пассивности на самом деле не существует, а если она имеет место, то скоро может потребоваться кто-то еще, кто бы мог делать вашу работу.

- Надеетесь ли вы, что ваши опытные программисты и без вас все сделают правильно, поскольку у вас просто нет времени на то, чтобы нянчиться с ними?

  Этот вопрос тесно связан с предыдущим — и опять же: если вы руководите квалифицированными людьми, то им может не требоваться ваше постоянное внимание. Только не забудьте о том, что я говорил относительно ожидания результатов без проведения проверок.

- Полагаете ли вы, что при приближении срока сдачи проекта все проблемы в конце концов решатся?

  Назовите это принятием желаемого за действительное или, на языке детей, ожиданием чуда. Как вы это ни назовете, полагаться на чудо опасно, и такая надежда обычно является результатом стресса. Многие видят в стрессе обыч-

ное жизненное явление, не всегда приводящее к пагубным эффектам. Стресс может быть мощным фактором, направляющим ваши усилия, если вы привыкли постоянно быть под напряжением и не сдаваться. Если вас не заводит ваша работа, возможно, вы выбрали не ту работу. Не бойтесь неудач, они когда-нибудь обязательно случаются; не бойтесь не успеть к сроку, это тоже когда-нибудь произойдет. Учитесь на своих ошибках; не научившись стойко переносить неудачи, вы никогда не будете знать, что делать с успехами.

- Считаете ли вы, что пользователи никогда не сделают ничего, что привело бы к программному сбою, просто потому, что у них не хватит ума для этого?

  Это тоже вопрос принятия желаемого за действительное. Из программистов получаются плохие бета-тестеры — поскольку мы подсознательно знаем, что определенные функции при проверке «грохнутся», мы и не пытаемся их проверять. В конечном счете ошибки всегда проявляются, так что не позволяйте поспешности влиять на качество. Как руководителю проекта вам стоит самому включиться в рабочий процесс на этапе альфа-тестирования, дабы выявить небрежности в программировании.

- Предпочитаете ли вы при управлении коллективом искать консенсус, даже если это требует массы времени и терпения?

  Я коснулся этого вопроса в конце предыдущей главы, а в этой подробно осветил роль времени и необходимость сохранять терпение, поэтому полагаю, что ответ очевиден. Если ваша интуиция вас иногда подводит, скажу прямо: консенсус должен быть вашей постоянной целью, и вы должны делать все, чтобы его достичь. Более подробное обсуждение этого вопроса вы можете найти в главах 5 и 6.

- Считаете ли вы электронную почту эффективным средством общения при работе над проектом?

  Ну конечно, она таковым не является, но если ваш коллектив пространственно разделен, она иногда остается единственно доступным средством общения. Совместное редактирование документа куда проще, чем запутанная переписка по электронной почте, однако электронная почта необычайно удобна, поэтому люди в первую очередь стремятся использовать именно ее. Только убедитесь, что у вас не накапливается непрочитанных сообщений, — лучше всего создать из них один документ, содержащий все, что имеет отношение к разработке.

- Согласны ли вы проговорить по телефону несколько часов кряду, только чтобы убедиться, что все идет как надо?

  Через это надо пройти. В вашей работе телефон — необходимое зло, если только все, с кем вы работаете, не находятся от вас дальше предела слышимости. В некоторых случаях телефон — ваше единственное средство проведения ежедневных проверок. Программы общения в режиме реального времени, наподобие ICQ, иногда заменяют телефон, но отнимают слишком много времени.

- Считаете ли вы, что с помощью комитетов невозможно выработать адекватные бизнес-требования?

Иногда комитеты и комиссии бесполезны, но обычно в большой и сложной организации они необходимы. Теперь, когда вы стали ответственным лицом, вам, возможно, предстоит быть членом или председателем большего числа разнообразных комитетов и комиссий, так что будет лучше, если вы научитесь их использовать. При удачном стечении обстоятельств комиссии могут быть мощным средством объединения интеллектуальных ресурсов, так что не надо недооценивать их значение. Как-никак, ни один человек не обладает всей необходимой информацией, а без знания бизнес-требований ваше программное обеспечение сгодится разве что для забавы.

- Считаете ли вы себя самым умным программистом в компании?

  Возможно, вы ответили на этот вопрос положительно. Причиной его задать было желание помочь вам побороть свое самомнение. Всегда, когда это возможно, старайтесь окружать себя теми, кто умнее. Никогда не обманывайте себя надеждой, что вы единственный, у кого есть все ответы. В конце концов, любое, даже самое оригинальное мышление — это просто химические процессы в мозгу, и чьи-то молекулы всегда могут быть лучше ваших.

- Чувствуете ли вы ревность и страх, когда смотрите на действительно хороший код, который написан не вами?

  Это вопрос-ловушка. Если вы гордитесь тем, что делаете, то вполне нормально чувствовать легкую ревность при виде чего-то, что сделано кем-то лучше. Под маской этой ревности обычно скрывается страх, что вас перехитрили или, что хуже, кто-то достиг чего-то, чего вы даже не в силах понять. Этот вид внутреннего эмоционального состояния «невовлеченности» часто поражает целые отделы. Осознайте эти ощущения и их причину и двигайтесь дальше. Ваша работа — качественно и в срок завершить проект, поэтому используйте все, что может помочь вам в этом.

- Делаете ли вы все возможное, чтобы успеть закончить проект в срок?

  Это возвращение к первому вопросу. Несоблюдение сроков может нанести убытки вашей компании и похоронить вашу карьеру. Нужны ли здесь еще какие-нибудь слова? Да. Лейтмотивом хорошего руководства разработкой программного обеспечения является прилежание, бдительность, внимание к деталям. Определение крайнего срока лежит в рамках этих составляющих руководства. Вы, может быть, вспомните, как Скотти из «Звездного пути» заслужил свою репутацию уникального работника[1], и примените схожий метод. Только не умножайте вашу исходную оценку времени, необходимого для завершения работ, на произвольное число, а назовите вашим людям какую-нибудь вымышленную дату, предшествующую дате, которую вы сообщите отделу тестирования. Чем больше вы будете практиковаться в оценке сроков, тем больших успехов вы достигнете в этом виде искусства.

Эти вопросы не были тестом — это были размышления о вашем положении начальника, позволившие выявить слабости, которые могли бы помешать вашему успеху.

---

[1] Скотти просто умножал оценочное время ремонта на 4.

КОШАЧЬИ РАЗБОРКИ — ПОХОРОННЫЙ МАРШ

Несколькими днями ранее этого прекрасного весеннего майского дня прошел последний срок сдачи, а работа над кодом еще не была завершена. Пение птиц и свежесть чистого неба напомнили Роджеру о прекрасном, он даже на секунду забыл о проблемах программного обеспечения встроенного процессора.

Поставив машину на свое вице-президентское место парковки в недавно построенном здании штаб-квартиры корпорации, Роджер обратил внимание, что на парковке не было машины Джеффа. Ну что же, еще один разговор о своевременности этим утром будет весьма кстати. Эти разговоры между Джеффом и Роджером за последние несколько месяцев случались часто. Они оба отдавали себе полный отчет в важности завершения нового программного обеспечения в срок. Казалось, что Джефф всегда будет отвечать, что он сожалеет и приложит максимум усилий, для того чтобы система заработала на этой неделе. Вице-президент по продажам уже подписал со многими компаниями по всей стране контракты на установку новой системы контроля и управления, которую Джефф и Роджер должны были наконец-то завершить. Президент дал ясно понять, что поскольку несколько крайних сроков уже прошли, работа обязательно должна быть закончена к этому новому крайнему сроку.

Роджер размышлял о том, что он мог бы сказать Джеффу такого, что было бы внове для него и могло бы заставить его наконец-то завершить работу. Он знал, что Джефф подолгу задерживается на работе вечерами и в выходные, — он звонил ему на всякий случай. Хоть бы код был написан не на C, тогда, может быть, Роджер смог бы помочь. Ну что ж, думал Роджер, он сделал все, что мог.

Прошло около тридцати минут рабочего дня, когда президент заглянул в просторный кабинет Роджера и сказал: «Спуститесь в мой кабинет на несколько минут. Вы мне нужны для небольшого разговора». Роджер поежился и подумал, что «небольшой разговор» — это обычная словесная выволочка от вышестоящего начальства. Когда Роджер присел напротив письменного стола руководителя компании, секретарь закрыла дверь в кабинет. Президент посмотрел в глаза Роджеру и произнес: «Вы догадываетесь, зачем я вас позвал?» Роджер не имел ни малейшего понятия, но пробормотал несколько слов о том, что сожалеет. Президент, как казалось, был в затруднении и просто сказал: «Собирайте ваши вещи. Вы уволены. Я вам месяц назад сказал, что к этому последнему сроку мы должны закончить. Чек с вашим выходным пособием получите у моего секретаря. Я хочу, чтобы вы покинули здание до полудня. До свидания».

Чему вы можете научиться из этой истории? Многому — в ней на каждом шагу намеки. Речь идет об очевидно успешной и богатой компании, отделом продаж в которой руководит очень напористый вице-президент. Он уже подписал контракты на систему, которая даже не была закончена. Другой же неудачливый вице-президент, ответственный за разработку программного обеспечения, на самом деле не имел необходимых технических навыков для того, чтобы понять, с какими затруднениями столкнулся программист Джефф. Вдобавок вместо того, чтобы сидеть вместе с Джеффом вечерами и в выходные и убедиться, что работа действительно сделана, он ограничивался телефонными «проверками» Джеффа.

Этот уволенный вице-президент совершил много ошибок. Во-первых, он не ощущал важности окончания работы в срок. Это проистекало из неуважения к деловым обязательствам. «Я сделал все, что мог», — замечательные последние слова, достаточно иррациональные и произнесенные исключительно для самоутверждения. Во-вторых, он был некомпетентен в данной проблеме. Этот вице-президент не знал C, но управлял программистом, который знал этот язык (но, очевидно, не очень хорошо). В-третьих, этот вице-президент не смог определить, в чем настоящая причина проблемы, — в программисте или в программном обеспечении. Он так и не выявил истинных причин надвигающегося бедствия. Итак, итоги вскрытия таковы: его уволили, и рассказанная история была реквиемом по делу, которое он делал.

Что дальше

На протяжении всей этой главы я держал перед вами зеркало (зеркало — метафора самопроверки и самонаблюдения). Глядеться в него было, возможно, неприятно и даже болезненно, но без боли вам не удастся вырасти до руководителя мужчин и женщин, разрабатывающих программное обеспечение в условиях жестких временных рамок. Есть ли секреты в управлении самим собой? Я не знаю ни одного, но точно знаю, что мое определение «сделаю все, что смогу» в контексте руководства программистами имеет другое значение. Бывает, когда фраза «я сделал, что смог» произносится под конец, в случае неудачи. Не думайте об этих словах в таком ключе. Эти слова означают приложение максимума усилий — их можно оценивать каждодневно, успех же иногда приходит только в самом конце. Так что если секрет и есть, то он таков: каждый день проверяйте себя как руководителя и старайтесь стать лучше уже на следующий день. Ваш перечень вопросов к самому себе мог бы выглядеть так:

1. Подвергаю ли я качество своего управления ежедневной оценке?
2. Действительно ли я с каждым днем руковожу все лучше или я постоянно откладываю совершенствование стиля и сути моего управления на потом?
3. Нравится ли мне то, что я делаю?
4. Теряю ли я попусту время при выполнении своих служебных обязанностей?
5. Оцениваю ли я свою производительность тем, сколько сделали мои подчиненные под моим руководством, или у меня есть ощущение, что сам я не сделал ничего?
6. Как мои слабости дали себя знать сегодня (по отношению ко мне самому или другим)?
7. Чему я научился сегодня, чтобы оставаться в курсе, чтобы быть осведомленным, чтобы углубить и расширить свои знания?

Этот перечень содержит семь пунктов; число семь древние считали совершенным. Вы не достигнете совершенства ни сегодня, ни назавтра, ни, возможно, в течение всей вашей жизни, но вы можете сделать максимально эффективными свои усилия, направленные на самосовершенствование, делая все, что нужно, ежедневно.

В следующей главе, в которой мы узнаем, как повести котов за собой, вам придется еще раз взглянуть в зеркало. Пускай самоанализ станет для вас островком безопасности, но не тайником. Выполняя вашу работу, вы должны предвидеть будущее и глядеть по сторонам, но при этом всегда необходимо иметь в виду тот факт, что становление вашего характера руководителя — процесс длиною в жизнь.

Глава 3

Как вести стаю за собой

Навыки руководителя нельзя выработать исключительно силой воли. Для того чтобы они появились, вы должны овладеть некоторыми, возможно, еще неизвестными вам принципами менеджмента. Эта глава как раз и посвящена ряду важных областей менеджмента, которые вам предстоит контролировать, ибо в противном случае они будут контролировать вас. Я намерен познакомить вас с теми аспектами вашей новой работы, которые непосредственно позволяют заставить двигаться ваших программистов в одном направлении — а, как я уже говорил, именно эта задача в процессе выпаса котов является для руководителя основной. Конкретнее, мы поговорим об организационном управлении, пронизывающем всю вашу рабочую деятельность. Я расскажу, что делать с раздражителями, которые имеют обыкновение постоянно отвлекать от работы. Этими навыками вы должны овладеть в обязательном порядке — вне зависимости от того, как они повлияют на ваши отношения с окружающими. Управление проектами — еще одно важное направление деятельности руководителя — рассматривается в этой главе в контексте остальных ваших обязанностей. Мы также поговорим о том, как формировать группы и как работать с персоналом, — теми людьми, которые определяют конкретный результат ваших усилий. Короче говоря, материал, содержащийся в этой главе, совершенно необходим для любого, кто хочет достичь совершенства в деле выпаса котов.

Как справиться с административными функциями

С тех пор как вы стали руководителем, ваше мировоззрение начало кардинально меняться. Будучи программистом, вы привыкли непринужденно общаться с себе подобными. Так вот, имейте в виду, что милые беседы по поводу того, как лучше организовать системное прерывание из-за ошибки, понемногу уходят в прошлое. Теперь вам придется обсуждать тонкости форматирования коммерческих требований, без чего вы просто не сможете составить спецификацию проекта. Конечно,

для написания такой документации существует огромное количество разных шаблонов, но проблема-то в том, что для решения этой и множества аналогичных задач, с которыми постоянно сталкивается руководитель, необходимо изменить стиль мышления.

Координация информационных потоков — это тот род деятельности руководителя, который требует отдельного рассмотрения. Как вы, я надеюсь, помните, согласно методике объектно-ориентированного (ОО) анализа и проектирования, для формулирования проектного решения и разрешения проблем, связанных с конструированием программного продукта, вы должны иметь в виду три перспективы. Если хотите освежить память, взгляните на табл. 3.1.

Таблица 3.1. Концепция объектно-ориентированного проектирования

Перспектива	Назначение
Бизнес	Анализ восприятия информации и процессов пользователем
Спецификация	Анализ публичных свойств интерфейса объекта
Реализация	Все внутренние детали объекта, обеспечивающие его функционирование

А теперь попробуйте привязать эту объектно-ориентированную концепцию к вашему повседневному общению с различными подразделениями компании, направленному на решение программных и управленческих задач. Вероятно, у вас получится своего рода программа интеллектуального анализа, похожая на показанную в табл. 3.2.

Таблица 3.2. Административный фильтр

Перспектива	Инструмент	Назначение
Бизнес	Глаза и уши	Сбор информации, необходимой для координирования программных задач
Спецификация	Организация	Систематическое отслеживание ожидаемых от вас результатов (в качестве единицы систематизации можно избрать, например, проект или технологическую область)
Реализация	Углубленный анализ	Дотошное выявление всех подробностей, составляющих ожидаемый результат

Предлагаемый мною административный фильтр в первую очередь призван помочь вам сориентироваться в информационной мгле[1], которая нас ежедневно окружает. Мгла эта состоит из электронных сообщений, телефонных звонков, технических условий, маркетинговых сведений, планов Microsoft на следующий год, вашей вечной головной боли в лице разгильдяя-программиста Джорджа и т. д. Другими словами, пытаясь вести свою стаю, вы ежедневно, если не ежечасно, попадаете в ситуацию информационной перегрузки. Все поступающие сведения на первый взгляд кажутся важными, однако без фильтра не обойтись. Смысл этого фильтра можно выразить одним словом (кстати, это еще один удачный термин из области объектно-ориентированного программирования):

ФОКУСИРОВКА

[1] Термин «информационная мгла» (data smog) я позаимствовал у Дэвида Шенка (David Shenk).

Помните ваш первый поход к врачу, когда еще ребенком вы выяснили, что должны носить очки? (Я задаю такой вопрос исходя из того, что вы — главный программист; следовательно, у вас большой опыт борьбы с мелочами жизни, а значит, вам, скорее всего, нужны очки.) Правда, замечательное ощущение, когда, надев линзы, вы понимаете, что теперь оптометрическая таблица вам нипочем? Это и означает умение сфокусироваться — отбросить все раздражающие факторы, которые неизменно сопровождают деятельность руководителя, и выявить первоочередные задачи.

Намотайте себе на ус: если уж у вас есть задача организовать производство программного продукта, остальные сведения, отвлекающие от этой задачи, независимо от того, насколько важными они кажутся, нужно отложить[1].

Если уж у вас есть задача организовать производство программного продукта, остальные сведения, отвлекающие от этой задачи, независимо от того, насколько важными они кажутся, нужно отложить.

Вот некоторые ситуации-раздражители, часто встречающиеся в повседневной деятельности руководителя.

- Кто-то из числа опытных пользователей вашего продукта придумал *замечательную* новую функцию и считает, что обязан рассказать вам о ней прямо сейчас.
- Сотрудник группы поддержки продукта в очередной раз сообщает вам о том, что количество записей в журнале ошибок с каждой неделей растет в геометрической прогрессии, и потому вы просто *обязаны* расставить в нем приоритеты.
- Проектировщик продукта испытывает страстное желание узнать, можно ли будет в обозримом будущем реализовать в коде желаемую функцию. По этой причине он пишет очередное письмо с настоятельной *просьбой* ответить на все предыдущие.

Обратите внимание на слова, выделенные курсивом: «замечательный», «обязан», «просьба». Смахивает на основной принцип подготовки вечерних теленовостей — «чем больше крови, тем лучше». Большинство раздражителей не так важны, как кажутся. Однако если их формулируют с помощью эпитетов типа «замечательный» и «обязан», а также с добавлением просьб, вы, скорее всего, не удержитесь, отвлечетесь от основной задачи и потеряете фокусировку. В этом-то и состоит проблема. Учитесь систематизировать те действия, которые вам рано или поздно придется выполнить, но не забывайте о том, что ваша главная задача — выпустить программный продукт. Иногда человеку, который обратился к вам с просьбой, полезно быстро ответить и, соответственно, дать знать, что вы его просьбу получили. Но при этом ответ должен выглядеть примерно так: «Я обязательно включу вашу проблему в график и позже рассмотрю, но в данный момент у меня очень много обязательств, и пока что я не могу этого сделать».

---

[1] В главе 1 я уже ссылался на труд Стивена Кави (Stephen Covey) под названием «The 7 Habits of Highly Effective People». Если у вас нет этой книги, идите в магазин, купите и прочитайте. Обязательно обратите внимание на то, что Кави думает по поводу организации времени и расстановки приоритетов.

Какое отношение, спросите вы, все эти разговоры о фокусировке имеют к ведению стаи в нужном направлении? Ну, если вы действительно хотите, чтобы коты за вами шли — а их, как известно, сложно заставить это сделать, — придется продемонстрировать им ваши лидерские качества. Не нужно *разглагольствовать* о лидерстве — вы обязаны показать программистам решимость выпустить общими усилиями качественный код. Естественно, руководитель, помимо прочего, должен учитывать все тонкости (в частности, те, что перечислены выше), которые помогают укомплектовать код. Фокусироваться — значит расставлять приоритеты среди тех сведений, которые претендуют на значимость наравне со сведениями, действительно значимыми для завершения текущих проектов.

Фокусироваться — значит расставлять приоритеты среди тех сведений, которые претендуют на значимость наравне со сведениями, действительно значимыми для завершения текущих проектов.

Предположим, например, такую ситуацию: в ходе написания кода вы замечаете в одном из объектов, который предполагаете расширить (или добавить интерфейс), некую аномалию. Искушение приняться за исправление такого объекта — по большому счету, попавшегося вам на глаза случайно — велико. Что сделает хороший программист? Он примет аномалию во внимание, но продолжит заниматься текущей процедурой. Вспомните, как просто потерять концентрацию при кодировании, отвечая на телефонный звонок. На то, чтобы в подобных ситуациях вернуться в колею (и восстановить в сознании логику кода), уходит битый час. В организационном управлении действует тот же принцип: либо вы решаете первоочередные задачи, либо теряете последние шансы укладываться в график. Ко времени, выделяемому на решение административных задач, нужно относиться так же трепетно, как и ко времени, выделяемому на кодирование.

Ко времени, выделяемому на решение административных задач, нужно относиться так же трепетно, как и ко времени, выделяемому на кодирование.

Как не отвлекаться на раздражители

Не позволяйте своему почтовому ящику влиять на ваш распорядок дня. С одной стороны, электронная почта — самое полезное в мире изобретение после хлеба в нарезке, но, с другой — есть в ней что-то от нечистого. Как средству оперативного обмена информацией ей нет равных; проблема в том, что в условиях географической рассредоточенности она становится основным носителем информационного потока. Позволив почте влиять на повседневные приоритеты, вы рискуете потерять концентрацию. Разрастание рамок проекта часто начинается с почтового сообщения, отправитель которого пытается уточнить коммерческие требования. Если дискуссия разрастается до трех сообщений, беритесь за телефонную трубку. Если звонок не решает проблему, попробуйте встретиться с собеседником лично. Если и это не принесет желаемого результата — бросайте все попытки его достичь.

Главное — чтобы электронная переписка не мешала решать задачи, включенные в график. Если в вашей компании имеется какая-то программа руководства проектами, опирайтесь на эту программу, собирая информацию по конкретным задачам; не загромождайте ее почтовым мусором, поскольку, если только не включать переписку в проектную документацию, она все равно, скорее всего, затеряется.

Еще один дьявольский инструмент уточнения — служба мгновенных сообщений. Пользоваться ею нужно с умом. Не тратьте на нее свое время впустую и не позволяйте ее значку на рабочем столе постоянно отвлекать вас от дела. Не пытайтесь с ее помощью проверять, находятся сотрудники на рабочих местах или нет, — никто не любит, когда за ним открыто наблюдают. В деле слежки нужно проявлять изобретательность — об этом, кстати, я говорил в главе 2 в подразделе «Бессмысленно ожидать чего-либо при отсутствии контроля» раздела «Естественный отбор и время».

Привыкая к решению организационных проблем, вы обнаружите, что раздражители влияют не только на вас — программисты подвержены им не меньше. Одна из главных ваших обязанностей заключается в том, чтобы приучить сотрудников концентрироваться на работе.

Одна из главных ваших обязанностей заключается в том, чтобы приучить сотрудников концентрироваться на работе.

Как справиться с этой задачей? Лучше всего еженедельно назначать всем программистам перечни задач, аннотируя их приоритетами и сроками завершения. Поскольку цикл разработки всегда отличается непостоянством, перечни эти могут меняться каждую неделю, а иногда — даже каждый день. Ответственность за составление и корректировку этих списков ложится, опять же, на вас[1]. Вы себе не представляете, сколько менеджеров считают программистов чуть ли не телепатами и пребывают в уверенности, что график работы они составляют с помощью интуиции. Таких убеждений придерживаются даже некоторые директора. Подобный подход регулярно приводит к тому, что вместо работы во имя пользователей программисты трудятся для программистов, а руководители обнаруживают свою бесполезность в контексте упрочения позиций компании.

Если вы унаследовали персонал от предыдущего руководителя, попросите каждого сотрудника в письменном виде сформулировать его текущие задачи. Это очень эффективный прием — вы не только узнаете, что программисты думают о своих обязанностях, но и составите представление о том, как руководство осуществлялось ранее. Просмотрев перечни задач, составленные самими сотрудниками, вы сможете оптимизировать их функции. Однажды приняв решение о руководстве методом перечней задач, вы не сможете от него отказаться. Все будут ждать новых заданий, и чем последовательнее вы себя проявите в деле их назначения, тем очевиднее станут ваши лидерские навыки. И еще один момент. Пере-

[1] Ряд предложений касательно программных продуктов, помогающих составлять для программистов перечни задач, содержится в главе 4. Но не торопитесь переходить к этому материалу — иначе пропустите все мои подготовительные пассажи о том, в какую путаницу вы рискуете попасть, если не призовете на помощь инструменты электронной организации рабочего процесса.

чень задач — это лишь первое звено в процессе ведения стаи в нужном направлении. По меньшей мере раз в неделю, а то и каждый день вам придется доносить до сотрудников дополнительные сведения, инструктировать их и помогать двигаться к намеченной цели, соблюдая при этом установленные временные ограничения.

Скорее всего, вам не удастся сильно продвинуться в вопросах физического размещения программистов, но все-таки при любой возможности требуйте, чтобы каждый сотрудник вместо отгороженной кабины имел отдельное помещение. Если руководство вашей компании больше думает не о продуктивности, а об арендной плате, у вас мало шансов — и, тем не менее, гните свою линию. Один из наиболее заметных раздражителей в деятельности программистов создают инженеры-технологи компаний с их квадратно-гнездовым мышлением. Фильм «Office Space», в котором, так сказать, «в естественном виде» изображены программисты в своих кубышках, должен стать обязательным для просмотра вредителями, планирующими офисное пространство. Факторы, влияющие на «отупение» сотрудников, на примере 32 346 компаний изучили Том Димарко (Tom DeMarco) и Тимоти Листер (Timothy Lister). На составленной ими диаграмме видна четкая обратная зависимость между степенью отупения сотрудников и объемом выделяемого каждому из них офисного пространства[1]. О чем это говорит? О том, что шум, отвлекающие факторы и все прочие побочные эффекты политики снижения затрат серьезно снижают продуктивность работы.

Так пусть ваши программисты работают дома — это один из лучших способов исключить раздражающие факторы и поднять продуктивность. Но и здесь не все так просто! Некоторым сотрудникам такой режим работы подходит лучше всего; другим полезно появляться на работе хотя бы раз в неделю; в любом случае, эффективной деятельности в режиме «на дому» достигают только самые дисциплинированные. Выбрав этот путь, вы будете вынуждены уделять довольно много времени проверкам. Опытным и надежным сотрудникам можно доверить работу на дому; что касается остальных — дайте им возможность попробовать, но обязательно проверяйте новые показатели продуктивности. Если отделения компании, в которой вы работаете, рассредоточены по всему земному шару, и у вас фактически нет выбора, будьте готовы к тому, что с прижатой к уху телефонной трубкой придется проводить по 10 часов в неделю, а от электронных сообщений не будет отбою. Схема работы на дому очень перспективна, но чтобы заставить ее приносить плоды, вы должны учесть все тонкости.

Когда проект разрастается

Я имею в виду классическое понятие *разрастания проекта*. Если аналитик бизнес-требований утверждает, что занимается уточнением рамок, знайте: он их *раздвигает*. Не важно, как этот процесс называть. Факт тот, что специфицировать и сконструировать программу, избежав разрастания требований, не удавалось еще никому. Объясняется это натурой человека — невозможно с первой попытки сформулировать все функции, которые предполагаемой программе предстоит выполнять.

---

[1] DeMarco and Lister, op. cit., p. 56.

Рассмотрим типичный процесс производства программного продукта. Вне зависимости от того, что вы предпочитаете — стремительный напор или постепенные итерации, — процесс этот всегда состоит из нескольких фиксированных этапов.

1. *Замысел*. У кого-то появляется блестящая идея.
2. *Специфицирование*. Куча людей пытаются описать эту идею.
3. *Проектирование*. Высокоинтеллектуальные товарищи решают, как сконструировать предполагаемый программный продукт.
4. *Конструирование*. Бессонными ночами и нескончаемыми днями программисты программируют.
5. *Тестирование*. Обнаруживается, что конечная реализация блестящей идеи не так хороша, как хотелось бы, или, еще хуже, что идея-то отнюдь не блестящая.
6. Все начинается заново со второго этапа, пока вы не почувствуете, что все нормально, всего хватает, или, наоборот, не придете к заключению, что идея, высказанная на первом этапе, ужасна и вам срочно требуется новый блестящий замысел (в последнем случае, опять же, все начинается со второго этапа).

А теперь подумайте: таким ли уж неожиданным станет разрастание рамок в контексте отраженного в этом списке реального сценария? Мне так не кажется, и вам тоже не должно так казаться. И тем не менее, если речь об этом зайдет, воя и зубовного скрежета программистов не избежать. Как заставить их поверить, что вы во всеоружии и ваша позиция правильна? Лучше всего поставить их перед фактом, что разрастание неизбежно, и научить их справляться с этой проблемой. Вы руководитель, и это — ваша обязанность. Не поддавайтесь соблазну разнести «крайних» в других отделах — этим вы не приблизите завершение работы и не исправите ситуацию.

В своей немного устаревшей, но, тем не менее, сохраняющей значимость книге под названием «Managing the Software Process» Уотс Хамфри (Watts Humphrey) сформулировал принцип, актуальный по сей день:

> «Когда программисты берутся за оценку объема кода реализации какой-либо функции, результаты неизменно оказываются заниженными. Этому есть множество возможных объяснений. В этом контексте следует понимать, что их оптимизм есть относительно прогнозируемая функция состояния проекта»[1].

На самом деле, разрастание рамок объясняется очень просто — сказать, сколько времени и сил уйдет на создание очередной сногсшибательной программы, вплоть до ее первого выпуска и критической оценки, не может никто. Многие программисты соглашаются с двумя соображениями относительно проводимых ими первоначальных оценок, согласующихся с принципом Хэмфри; я сформулирую их в виде аксиом:

- любой процесс продлится дольше, чем вы надеетесь;
- всегда появляется что-то, о чем вы не подумали.

Вооружившись этими аксиомами и введенным Хэмфри понятием «состояния проекта», старайтесь контролировать разрастание и обязательно убедите ваших

---

[1] Watts S. Humphrey, Managing the Software Process (New York: Addison-Wesley, 1989), p. 93.

подчиненных в том, что вы человек проницательный, поскольку предсказывали такую возможность и учли ее в процессе тщательного планирования. «Тщательное планирование»! Звучит замечательно, но что же это означает в контексте выпаса котов? А вот что. Вернитесь к главам 1 и 2, еще раз пробегитесь по изложенным в них принципам и попытайтесь сделать их неотъемлемыми элементами своего характера. Помните, что лидерство происходит от сердца, а не от ума[1].

Конечно, и для мозга найдется работа — в частности, ему предстоит разработать методы контроля над разрастанием рамок проекта. Давайте рассмотрим типичный план проекта. Предположим, что, исходя из заданного набора требований, вы должны разработать проектное решение с расчетом на его последующую реализацию программными средствами. Элементарный, но наивный план иллюстрирует табл. 3.3.

Таблица 3.3. Нереалистичный план проекта

Задача	Время выполнения (произвольные интервалы)
Анализ требований	A
Создание проектного решения	B
Реализация проектного решения	C
Тестирование программного обеспечения	D
Исправление ошибок	E
Развертывание программного обеспечения	F

Руководствуясь таким планом, вы рискуете нарваться на кучу неприятностей. Будучи глубоко убеждены, что конечная дата сдачи проекта будет равняться сумме временных интервалов A + B + C + D + E + F, вы немало удивитесь, обнаружив, что это совершенно не так.

Рассмотрим более реалистичный план, показанный в табл. 3.4.

Таблица 3.4. Реалистичный план проекта

Задача	Время выполнения (произвольные интервалы)
Анализ требований	A
Обсуждение результатов анализа с сотрудниками отдела	B
Создание проектного решения	C
Макетирование проектного решения	D
Оценка макетов	E
Пересмотр проектного решения	F
Реализация высокоуровневых объектов проектного решения	G
Тестирование высокоуровневой интеграции	H
Оценка системы на предмет соответствия требованиям	I
Создание компонентов системы	J
Интеграция и тестирование компонентов	K
Повторная оценка системы на предмет соответствия требованиям	L

[1] Вспомните, что говорил Йода: «Сила Джедая есть результат его собственных усилий». У нас то же самое.

Задача	Время выполнения (произвольные интервалы)
Тестирование комплектной системы	M
Исправление неисправностей системы в преддверии альфа-тестирования	N
Начало альфа-тестирования	O
Исправление ошибок, выявленных на этапе альфа-тестирования	P
Начало бета-тестирования	Q
Разработка стратегии развертывания	R
Исправление ошибок, выявленных на этапе бета-тестирования	S
Тестирование стратегии развертывания	T
Тестирование конечного продукта	U
Развертывание программного обеспечения	V

Проводить арифметические операции я, пожалуй, не буду, поскольку в таблице мне еле хватило букв латинского алфавита. В общем, вы все поняли. Помните также, что приведенный план не учитывает тех индивидуальных этапов, которые обусловливаются особенностями проекта, равно как и зависимости между различными этапами цикла разработки. В то же время он успешно иллюстрирует причины разрастания рамок проектов, позволившие мне сформулировать две вышеупомянутые аксиомы: это, во-первых, недостаточный анализ подробностей, и, во-вторых, излишне оптимистические оценки длительности конструирования программных продуктов. Чем больше опыта вы наработаете в деле выявления подобных деталей, тем точнее вы сможете оценить время работы над проектом и тем больше вы будете убеждаться в том, что разрастание рамок проекта происходит в результате недоработок специалистов, отвечающих за планирование.

По большей части, разрастание рамок проекта происходит по причине недоработок специалистов, отвечающих за планирование.

Каков механизм совершенствования навыков временной оценки? Он очень прост — поначалу вам предстоит неоднократно нарушать утвержденные графики сдачи проектов, но, в конце концов, вы научитесь им соответствовать. Практика эта довольно нервозная и даже угрожающая вашему карьерному росту. Возможно, это не лучший способ самосовершенствования, но что можно утверждать со всей определенностью, так это то, что в области управления проектом опыт играет огромную роль. А чтобы ускорить обучение, почитывайте анекдоты, относящиеся к руководству проектами. В своей леденящей душу коллекции очерков о неудавшихся программных проектах Роберт Гласс (Robert Glass) составил список наиболее распространенных «программных катастроф», который я привожу в порядке снижения значимости[1].

1. Неадекватное специфицирование задач проекта (51 %).
2. Неудовлетворительные планирование и оценка (48 %).

[1] Robert L. Glass, Software Runaways (Upper Saddle River, NJ: Prentice Hall, 1998), p. 20.

3. Применение новой для данной компании технологии (45 %).
4. Негодная/отсутствующая методология руководства проектом (42 %).
5. Нехватка ведущих специалистов группы (42 %).
6. Срыв договоренностей производителями аппаратного/программного обеспечения (42 %).

Процентные показатели, приведенные для каждой из вышеупомянутых причин несостоятельности, выведены Глассом в ходе специального исследования с целью выявить основную причину выхода процесса разработки программных продуктов из-под контроля. Я очень рекомендую ознакомиться с этой и другими подобными работами[1]. В результате вы поднимете процесс обучения на более осмысленный уровень, получите определенное представление о реалистичном планировании проекта, овладеете методикой контроля над разрастанием рамок проекта.

Как объединить усилия тех, кто гуляет сам по себе

Речь не о кошках, а о программистах. О том, что они блуждают в потемках, можно судить по характеру их кода, — он начинает походить на огромный памятник их гениальности. Практика регулярного критического обзора кода помогает сносить подобные монументы еще до того, как самовлюбленные хозяева их окончательно возведут. Более подробно мы побеседуем об этом явлении в главе 6, полностью посвященной техническому руководству. Пока что запомните один замечательный принцип: творчество бесценно, а вот практичный и удобный в сопровождении код можно не только оценить, но и продать. В ваши обязанности как руководителя входит координация деятельности программистов, направленная на достижение максимальной функциональности за счет минимального объема кода. Усложнение — это то, с чем вам предстоит вести постоянную борьбу. К понятности кода придется идти мелкими шажками — благо, скорее всего, вам предстоит бороться со смертными грехами программистов, такими как[2]:

- недостаточное функциональное разделение при создании объектов и стыковке логических уровней;
- непродуманные интерфейсы объектов;
- чрезмерная взаимозависимость объектов;
- пристрастие к усложнению внутреннего устройства объектов.

В попытках исцелить молодых и неопытных программистов проявляйте такт. Упираются рогом? Ради бога — пусть некоторое время поучатся на собственных ошибках; при этом не забывайте регулярно показывать им правильное направление. Естественно, прежде чем допустить их код до компиляции, предложите свои коррективы. Мы, программисты (впрочем, как и все человеческие существа), имеем обыкновение совершать ошибки, лицезреть их последствия и только после

---

[1] См. также Robert L. Glass, ComputingFailure.com (Upper Saddle River, NJ: Prentice Hall, 2001).

[2] Это лишь немногие грехи. Существует множество альтернативных перечней. Добротный пример см. в издании William H. Brown et al, AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis (New York: John Wiley & Sons, 1998).

этого искать лучшие пути достижения тех же целей — так мы учимся. При том условии, что допущенные ошибки будут исправлены до альфа-тестирования, в них нет ничего страшного.

В отличие от плодов творчества, которое бесценно, практичный и удобный в сопровождении код можно не только оценить, но и продать.

Опасность!

Если навыки программистов, которыми вы руководите, отличаются от ваших собственных, скорее всего, возникнут трудности. Ну, скажем, вы специализируетесь на VB, а сотрудники ваши делятся на приверженцев ASP и тех, кто пишет на традиционных языках. В таких условиях проводить критические обзоры кода вам будет непросто, а это уже порождает некоторую опасность. Возможно, окончательно определиться с тем, насколько качественно поработали ваши сотрудники, удастся только на этапе итогового тестирования. Скорее всего, вам придется основательно взяться за изучение новых языков и методов. Лишь в этом случае вы сможете выполнять свои обязанности эффективно и не станете предметом насмешек для программистов, которые, решая поставленные вами задачи, задействуют в своих решениях кучу незнакомых понятий.

Если хотите снизить риск, связанный с попытками управлять кодом, который вы не понимаете, заведите себе помощника, разбирающегося в неподвластных вам языках. Проведение критических обзоров кода предполагает элементарное умение его, скажем так, читать. В своем классическом труде о том роде человеческой деятельности, который мы называем кодированием, Джеральд Вейнберг (Gerald Weinberg) пишет:

> «Несколько лет назад, когда язык COBOL еще называли будущим программирования, очень активно обсуждался вопрос о том, умеют ли руководители читать программы. По прошествии времени кажется совершенно очевидным, что все эти разговоры были нацелены лишь на выбивание средств из руководителей, стремившихся свести свою зависимость от программистов к минимуму. На самом деле никто не верил, что руководители умеют читать программы. И с какой стати они должны это делать? Ведь даже программисты не читают программы!»[1].

Естественно, дальше по тексту автор утверждает, что руководитель обязан уметь читать программы, и даже если не умеет, должен обязательно привлечь кого-то, кто с этой задачей справляется. В цитате упомянута зависимость от программистов — это та самая вещь, от которой вы всеми силами будете стараться избавиться. Найдите среди своих сотрудников кого-то, кто смог бы помочь вам свести к минимуму ущерб от недостатка знаний в тех или иных областях. Время от времени вам придется сталкиваться с устаревшими технологиями, поскольку для решения отдельных проблем они вполне адекватны. У руководителя есть еще одна цель: состоит она в том, чтобы разрушать новые (и даже старые) границы,

[1] Gerald M. Weinberg, The Psychology of Computer Programming: Silver Anniversary Edition (New York: Dorset House Publishing, 1998), p. 5.

открывая при этом новые пути в программировании, — те, по которым ранее никто не додумался пройти. Имейте в виду, что невежество очень опасно. Самое страшное — это когда человек упорствует в своем невежестве, выдавая его за принцип; такое поведение иначе как глупостью не назовешь.

Готовность к постоянному углублению своих знаний есть одна из самых выгодных черт руководителя. Если это не про вас, советую переосмыслить ваши рабочие обязанности или поискать новую работу. Дело в том, что в условиях постоянного изменения технологий вы должны быстро схватывать новые идеи и трезво оценивать их достоинства. Возможно, вы и не станете экспертом, но ориентироваться в профессиональном сленге и специальных понятиях совершенно необходимо — в противном случае вы просто не сможете успешно проводить своих сотрудников через технологические джунгли.

Готовность к постоянному углублению своих знаний есть одна из самых выгодных черт руководителя. Если это не про вас, советую переосмыслить свои обязанности или поискать новую работу. Дело в том, что в условиях постоянного изменения технологий вы должны быстро схватывать новые идеи и трезво оценивать их достоинства.

Как сформировать команду и как ее поддерживать

Еще одна неотъемлемая черта хорошего руководителя — это умение находить хороших сотрудников. Имейте в виду, что подчиненные судят о ваших лидерских качествах исходя из того, каких новых специалистов вы приводите и от каких избавляетесь. Вскоре вы, вероятно, обнаружите, что вопросы найма, увольнения и вознаграждения из всех областей деятельности руководителя наиболее сложные. С другой стороны, действия подобного рода обогащают ваш опыт как специалиста, поскольку люди — это основной элемент процесса разработки программных средств. Чем лучше ваши сотрудники, тем более грандиозных целей вы сможете достичь. Если же вам придется иметь дело с персоналом уровня «ниже среднего», то большую часть рабочего времени вы будете вынуждены посвящать решению тех задач, которые совершенно не способствуют достижению поставленных целей.

Как нанимать сотрудников

В процессе поиска новых сотрудников следует проявлять здравомыслие — в конце концов, вам самому придется сосуществовать с новыми людьми и помогать им адаптироваться в команде. На это уходит много сил и времени, однако взвешенный подход к вопросам найма гарантирует решение задач. Далее я привожу ряд факторов, которые обязательно нужно учитывать при поиске новых программистов.

- Заставьте кандидата выполнить тестовое задание. Поставьте перед потенциальным сотрудником задачу, которую он должен будет либо решить сразу, либо взять на дом и решить к установленному сроку.
- Обязательно проводите устную проверку навыков кандидата. Если у него есть сертификаты, протестируйте его по одному из них и оцените полученные кандидатом знания, а заодно и его способность решать задачи в стрессовой ситуации.

- Если вам удастся убедить кандидата в том, что ему следует пройти сетевой тест оценки личности, зная, что возможности такого рода проверки весьма ограничены, значит — действуйте.[1]
- Составьте письменное описание предполагаемых функций кандидата и попросите его ознакомиться с ними непосредственно во время интервью. В отсутствие такого списка кандидат может подумать, что вы не знаете, что вам нужно, и будет говорить лишь то, что вы ожидаете от него услышать.
- Не ограничивайтесь одним интервью. Если возможно, попросите одного из ваших лучших и доверенных подчиненных провести с кандидатом еще одну беседу. В то же время не позволяйте потенциальному сотруднику общаться со всем персоналом — таким способом вы не добьетесь взвешенного решения.

Если вам доведется нанимать консультанта на ограниченный срок (а по-другому консультанты не работают), проблема вхождения в команду, по большому счету, отпадет; тем не менее некоторые из вышеприведенных инструкций подходят и для таких случаев. Остерегайтесь консультантов, которые демонстрируют желание остаться у вас на постоянную работу. Преданные сотрудники, успевшие поучаствовать в достижениях компании, в любом случае лучше, чем консультанты, которые думают в основном о том, сколько им будут платить за час работы и как долго они смогут на ней продержаться.

В защиту консультантов скажу, что если в вашей команде не находится человека, способного принять в связи с какой-то проблемой компетентное решение, а такой человек вам необходим, смело обращайтесь к консультанту. Попытайтесь сделать так, чтобы он как можно быстрее поделился своими знаниями с постоянными сотрудниками отдела. Хороший консультант должен быть нацелен на конструктивное взаимодействие с вашими сотрудниками, анализ предложенной вами проблемы, предложение идей, о которых вы по тем или иным причинам не подумали, и завершить на этом свою миссию. Иногда хорошие отношения с консультантом позволяют предложить ему постоянную позицию в команде. Во многих случаях такая возможность исключается договором о консультировании, однако если таких ограничений нет, не упустите возможность проверить, как консультант взаимодействует с сотрудниками, и только после этого принимайте окончательное решение.

Обычно консультанты не заинтересованы в том, чтобы передавать клиентам свои знания, — ведь это гарантия их дальнейшего пребывания на своей позиции. В таких случаях сотрудникам приходится проводить дизассемблирование кода, созданного консультантом, — это единственный способ разобраться в предложенном им решении. Подобная практика довольно распространена — в конце концов, многие из нас учатся на примере мастеров программирования, результаты работы которых можно встретить в книгах, существующих проектах, Интернете. Учиться нужно при любой возможности и на любом примере. Старайтесь только не пасть жертвой консультанта, оставившего вас, выражаясь словами Черчилля, лицом к лицу с загадкой, которая завернута в головоломку, а головоломка — в ребус.[2]

---

[1] Достойные тесты оценки личности есть на сайте http://www.advisorteam.com.

[2] Это знаменитая фраза Черчилля — так он охарактеризовал Советский Союз перед началом Второй мировой войны. Если будущее вашего продукта зависит от результатов работы скрытного консультанта, будьте готовы к тому, что на его сопровождение уйдет много усилий.

КОШАЧЬИ РАЗБОРКИ — БЛЮЗ ОДИНОКОГО ФЕРЗЯ

Фрэнку жутко хотелось нанять нового сотрудника. Лишь недавно поднявшись до руководителя группы программистов, он горел желанием внести в ее кадровый состав свой посильный вклад. Преклоняясь перед программистами из страны — королевы шахмат, он нашел первое резюме с устраивавшими его характеристиками и назначил интервью. Итак, перед ним сидел Алекс — человек, который, как Фрэнк вскорости узнал, мог играть в шахматы без ферзя и при этом побить любого противника. Фрэнка совершенно не смущали напряженные отношения Алекса с английским, ибо он был твердо уверен, что этот программист способен создавать сногсшибательный код. Итак, Фрэнк нанял Алекса, и с этого момента началась многолетняя история их занимательных профессиональных и личных отношений.

В большинстве проектов Алекс проявлял себя исключительно творческим человеком. Он ухитрился организовать полноценный графический интерфейс пользователя на черно-белом жидкокристаллическом экране, на котором можно было вывести всего четыре строки по 80 символов в каждой. В общем, остальные сотрудники группы остались под впечатлением. В свободное время он смастерил для разрабатываемых продуктов внутрисхемный эмулятор, с помощью которого значительно сократил продолжительность отладки. Тем не менее с английским у Алекса так и не сложилось, и хотя Фрэнк помог ему получить вид на жительство, нового сотрудника, по большому счету, так и не приняли в свой круг его коллеги.

Понимая, что Алекс находится в изоляции, Фрэнк стал все чаще назначать его на проекты, рамки которых не предполагали участия нескольких разработчиков. В течение нескольких лет Фрэнк не мог нарадоваться результатам Алекса. Только вот остальные сотрудники, которым приходилось сопровождать созданный Алексом код и которые по этой причине постоянно на него жаловались, надоедали Алексу все больше. Он объяснял такое отношение профессиональной завистью и потому с легкостью отвергал все возмущенные высказывания.

Через некоторое время Фрэнк ушел из родного банка и пустился в свободный полет. Еще через несколько лет он пересекся с неким Бобом — владельцем конкурирующего банка, который нанял Алекса по рекомендации Фрэнка. Боб тут же разразился ужасными историями о том, как трудно ему пришлось с Алексом, которого в конечном итоге уволили. Дело в том, что никто не мог справиться с расширением продуктов, которые Алекс штамповал с большим апломбом и претензией на стиль.

Что стало с Алексом, неизвестно. Обе компании, в которых ему довелось работать, потерпели значительные убытки, так как созданные им программные продукты так и не продвинулись дальше первой версии. Парадоксально то, что Алекс в этом не виноват. Все началось с ошибки Фрэнка, который предпочел «гениальные» программные монументы возможности работы в команде и потребности в практичном и допускающем сопровождение коде. Нанимая к себе в группу Алекса, Фрэнк поступил необдуманно — в тот момент он делал то, что нужно было ему самому, и этот поступок не имел никакого отношения к перспективам деятельности его компании. Поддерживая изоляцию Алекса от других сотрудников, Фрэнк привел подведомственную ему группу к расколу — и все из-за того, что, по его мнению, выдающиеся способности могут компенсировать нечеткие проектные решения. Делайте выводы. Нанимать умных людей недостаточно — их нужно нанимать с умом!

Как увольнять сотрудников

Увольнение — это обратная сторона найма. Она в не меньшей степени выражает ваши лидерские качества. Один-единственный некомпетентный или просто про-

блемный сотрудник способен разрушить всю команду. При отсутствии ограничений правового характера не стесняйтесь — если человек не попытался исправиться, смело увольняйте его. Если условия найма предполагают прохождение кандидатом испытательного срока, в продолжение которого он показал себя неспособным к решению поставленных задач, обязательно воспользуйтесь законной возможностью его выгнать и действуйте, пока не поздно. Чем дольше в вашем окружении останется плохой программист, тем хуже для вашей команды, поскольку сотрудники рискуют заразиться его вредными привычками. Кроме того, не забывайте, что судят о вас в том числе и по тому, как вы обращаетесь с людьми, которые, по всеобщему мнению, должны уйти. Бездействие — тоже действие, причем отрицательного характера. Не забывайте, что с тем, кто боится принимать решения, когда ситуация совершенно ясна, случаются большие неприятности.

Если вы подумываете о том, чтобы уволить какого-то сотрудника, спланируйте свои действия заранее. Зафиксируйте в письменном виде те неверные действия, которые предпринял сотрудник, и те трудности, причиной которых он стал. Предложите ему один, последний, шанс исправиться. Некоторые специалисты утверждают, что одного шанса мало, но, по моему опыту, даже две таких возможности — это непозволительная роскошь. От вас как от человека, курирующего исправительный период, потребуется слишком много сил; остальным же сотрудникам, если им придется с ним работать, придется очень трудно. Тем не менее просчитайте возможные последствия увольнения для компании в целом. Попросите своего начальника оценить принятое вами решение. Какие проблемы, связанные с неразглашением конфиденциальных сведений, могут появиться у компании в связи с увольнением? Быть может, программист, от которого вы намерены избавиться, обладает обширными знаниями и просто не успел их проявить. На эти и другие связанные с увольнением вопросы вам придется ответить.

Кроме того, обязательно обдумайте влияние сокращения отдельного программиста на всех остальных сотрудников отдела. Им, скорее всего, захочется узнать, по какой причине одному из их друзей дали «от ворот поворот». Впрочем, не стоит сообщать им все подробности — достаточно сказать, что уволенный специалист демонстрировал недостаточную продуктивность, в связи с чем его последующее пребывание на должности противоречит интересам отдела. Помните, что увольнение, помимо прочего, приводит к положительному эффекту — вы избавляетесь от проблемных сотрудников, а все остальные понимают, что присутствия подобных деятелей в отделе вы не потерпите. Иногда подать сотрудникам такой сигнал совершенно нелишне.

Денежное поощрение и продвижение сотрудников по службе

Вопросов денежного вознаграждения я касался в главе 1. Скорее всего, именно в этой области управления вы столкнетесь с наибольшими трудностями. Некоторые предпочитают награждать новыми должностями, но никому пока что не удавалось избежать практики денежного поощрения. Какую тактику избрать вам? Одно из возможных решений нашло в 1960-х годах руководство Bell Labs — все исследователи в этой лаборатории получали должность «научно-технического сотрудника», и более высоких почестей, нежели вхождение в эту закрытую группу,

в компании просто не существовало. Такая схема работает лишь в том случае, если штат вашей организации набран исключительно из высококвалифицированных сотрудников. В сегодняшних условиях обращение к ней зачастую не оправдано — с одной стороны, из-за того, что должности в компании устанавливаете не вы, с другой — из-за стремления отдельных специалистов к звучным званиям. Есть такая закономерность: чем моложе сотрудник, тем большее рвение он проявляет в погоне за высокой должностью. У людей постарше другие приоритеты — им нужна интересная работа и хорошие деньги. Если сотрудник хочет всего вместе значит — он либо действительно этого заслуживает, либо несколько неадекватно оценивает свои способности. Поскольку интересную работу получают большинство программистов, вам придется довольствоваться вариантами вознаграждения должностями и деньгами, основываясь при принятии решений на достоинствах и опыте конкретного человека.

Продвигая сотрудников ввысь по должностной лестнице, делайте это с осторожностью. Обязательно оценивайте, по силам ли им новые обязанности. Многие программисты, да и технари в целом, готовы всю жизнь довольствоваться интересной и творческой работой, даже не задумываясь о том, чтобы принять на себя руководящие функции. Учитесь определять максимальный доступный сотруднику уровень обязанностей. Возможно, программист, за которым вы наблюдаете, сможет в будущем достичь большего, но обычно для этого человек должен в течение определенного времени привыкать к выполняемому им уровню обязанностей. Как только вы поймете, что со своими нынешними функциями он справляется вполне успешно, можете поднять его еще на ступеньку вверх. Процесс продвижения сотрудников, как правило, носит итерационный характер — подобно определению коммерческих требований и макетированию. Прежде чем нагружать сотрудника более серьезными функциями, необходимо убедиться в том, что он справляется с текущими. Некоторые считают, что чем больше обязанностей выполняешь, тем больше денег получаешь. В некотором отношении это так. В то же время следует помнить, что жалование нужно увеличивать в ответ на повышение продуктивности и эффективности выполняемой работы, а не просто за расширение рамок проекта и распределение обязанностей между подчиненными.

Некоторые компании выстраивают четкую иерархию технических сотрудников, поднимаясь по которой любой квалифицированный специалист может получать зарплату, сопоставимую с окладами высшего руководящего звена. Если вы работаете в такой организации — что ж, здорово! Эта политика исключает стремление технических специалистов получить руководящие должности лишь для того, чтобы заработать больше денег. В таком случае не удивляйтесь, что некоторые специалисты будут получать больше, чем вы. В организациях, где такая система практикуется, высокие зарплаты выдаются в основном тем сотрудникам, которые помогли компании удержаться на плаву в трудные дни. Действительно ли подобных передовиков, работающих в вашей компании, стоит оценивать финансово выше, чем вас? Вполне может быть, что стоит, и вы должны ценить таких людей, радоваться, что они есть в вашем распоряжении. В своей книге, восхваляющей искусство кодирования, Пит Макбрин (Pete McBreen) поднимает вопрос о реальной финансовой оценке ведущих разработчиков:

«Возможно, они стоят значительно большего, чем получают в данный момент. Вероятно, они должны получать в пять или даже десять раз больше, чем сред-

нестатистический разработчик... Чего на самом деле стоит человек, который «спас» проект? Для того чтобы ответить на этот вопрос, имеет смысл оценить последствия, которые могли бы наступить, если бы такой сотрудник перешел в другую компанию»[1].

Оценивайте своих сотрудников по достоинству. Платите им столько, сколько они заслуживают. Не нужно чрезмерно экономить, иначе вы можете их потерять. Скорее всего, вы слышали о классическом треугольнике «дешево — быстро — качественно» применительно к процессу разработки программных средств. Так вот, из этих трех качеств сочетаться могут только любые два. Если вы придерживаетесь практики привлечения низкооплачиваемых сотрудников, то результат получите соответствующий — дешевое (во всех смыслах) программное обеспечение. Если хотите получить качественный конечный продукт, имейте в виду, что за качество нужно платить.

Как готовить преемника

Что?! Я только что получил эту работу и воспитывать себе смену не собираюсь! Этот ход мысли не отличается проницательностью. Никогда не забывайте о «факторе падающего кирпича». А вдруг вас приберет Бог или случится какое-то несчастье? Кто вас заменит (или сменит)? Как отдел продолжит работу? В некоторых ситуациях кому-то все равно придется исполнять ваши обязанности — для этого не обязательно попадать в беду. Выстраивая свой стратегический план, вы должны постоянно учитывать такие возможности и искать людей, способных время от времени подменять вас. Это помогает, с одной стороны, выявить сотрудников, заслуживающих более широкого круга обязанностей, с другой — определить набор высокоуровневых организационных задач, которые можно с уверенностью делегировать другим.

С поиском преемника тесно связана проблема углубления познаний сотрудников и формирования своеобразной «скамейки запасных». Если каждый из ваших программистов специализируется на определенном проекте, в случае его отсутствия на рабочем месте вас ожидают серьезные неприятности.

В подготовке спортсменов широко применяется методика обучения смежным видам; так почему бы не опробовать ее на программистах? Согласно распространенному мнению, зрелость в профессии программиста наступает тогда, когда он овладевает вторым языком. Таким образом, нужно стремиться к повышению компетенции персонала, с тем чтобы, переходя от одного проекта к другому, специалисты не теряли темп работы и концентрацию. С вашей стороны для достижения этой цели потребуется серьезное планирование, но, по большому счету, проблема эта не нова — думать о том, как распределить задачи между программистами, руководитель должен постоянно. Некоторые легко переходят от задачи к задаче, у других такой мобильности не получается. Изучайте сильные и слабые стороны своих подчиненных и пользуйтесь этими знаниями в интересах общего дела.

Ну хватит уже!

Полагаю, именно так вы сейчас мыслите. Действительно, быть руководителем непросто — с вашей стороны требуются определенные усилия. Организационные

---

[1] Pete McBreen, Software Craftsmanship (New York: Addison-Wesley, 2001), p. 61.

вопросы, которые вам предстоит решать, изрядно выматывают нервы. Впрочем, эффективность ведения стаи в конечном итоге дает свои плоды, и немалые. В любом случае команда значительно более продуктивна, чем отдельный человек, — каким бы талантливым и работоспособным он ни был. Все те области деятельности руководителя, которых я касался в этой главе, имеют отношение к вашим обязанностям по эффективному кадровому обеспечению. Именно в этом и состоит цель руководителя — сформировать группу сотрудников, способную последовательно проводить в жизнь крупные проекты. Мощь и энергия группы накапливается путем совершенствования руководящих навыков.

Если попробовать свести содержание этой главы к нескольким четким и удобоваримым принципам, получится, что руководитель:

- расставляет приоритеты и борется с раздражителями (фокусируется на поставленных задачах);
- совершенствует свои навыки в области руководства проектами и прорабатывает все их детали;
- пресекает низкое качество кодирования, пока оно не пустило в проекте корни;
- стремясь по достоинству оценивать технологические новинки, учится быстро усваивать неизвестную информацию;
- благоразумно относится к кадровому обеспечению и понимает, что именно от людей зависит конечный успех проекта.

Посмотрите — я ухитрился уместить смысл всей главы в нескольких фразах. Следуйте моему примеру: вместо того чтобы скверно делать множество дел, сконцентрируйтесь на нескольких задачах, но доведите их решения до совершенства. Многозадачность — свойство микропроцессоров, но никак не людей.

Что дальше

Вам еще многое предстоит. Если бы руководить было так просто, этим занимались бы все без исключения. В следующей главе мы рассмотрим еще один аспект лидерства, конкретнее — разберемся с тем, как усовершенствовать ваши организационные навыки. А пока что отдохните. Заканчивайте с чтением на сегодня. Займитесь чем-нибудь приятным, уважьте свой ум и сердце.

Когда после этого вы с новыми силами возьметесь за книгу, мы разберемся, как организовать вашу административную деятельность так, чтобы она принесла успех. Можно сказать и так: с помощью следующей главы вы сможете превратить теорию в практику. Для четкой постановки задач и их решения зачастую требуется пройти довольно болезненный этап изменения своих привычек. Прежде чем менять свои действия, нужно исправить мышление; иначе без активных действий вы рискуете превратиться в руководителя-теоретика. В процессе выпаса котов вам предстоит столкнуться с жесткими реалиями, набить себе шишек, поучиться на собственных ошибках и, наконец, систематизировать тот хаос, который зачастую царит среди находящихся в свободном полете программистов. Оставайтесь со мной, и у вас все получится.

Глава 4

Как организовать успех

Первоочередная задача и желательный результат деятельности любого хорошего менеджера заключается в обеспечении условий для достижения успеха подведомственной ему группой разработчиков. Впрочем, некоторые специалисты, занимающие ответственные должности, позиционируют себя «всего лишь» менеджерами и абстрагируются от лидерских аспектов деятельности. Если и вы придерживаетесь того же мнения, значит, вы, возможно, перегружены повседневными задачами и вам просто не хватает времени для стратегического планирования развития своей группы. Поймите меня правильно — менеджер действительно должен следить за тем, чтобы все текущие задачи своевременно исполнялись; лидер же, помимо решения повседневных задач по повышению продуктивности сотрудников, концентрируется на стратегических задачах своей группы и не ограничивает себя соблюдением контрольных сроков. Менеджеры иногда погрязают в хитросплетениях своей работы; лидеры, напротив, всегда стремятся разработать новые приемы, позволяющие подчиненным брать все более высокие планки. Этот своеобразный конфликт между функциями менеджера и качествами лидера напоминает хождение по канату — в том смысле, что нам все время приходится уравновешивать две в равной степени значимые для нашей профессии роли.

Если бы перед нами стояла проблема однозначного выбора, наверное, можно было бы сказать, что лидерство важнее руководства. С другой стороны, если не уделять внимания не слишком связанным, на первый взгляд, с практикой аспектам руководства (в частности, организационным навыкам), времени на реализацию лидерских качеств просто не останется. Чем более организованным вы станете — как с личностной, так и с профессиональной точки зрения, — тем легче будет поддерживать баланс между руководством и лидерством. Организованный менеджер сам создает для себя условия, в которых лидерские качества начинают расцветать.

Организованный менеджер сам создает для себя условия, в которых его лидерские качества начинают расцветать.

Нарабатывая личностные и профессиональные организационные навыки, никогда не забывайте о конечной цели, — она состоит в том, чтобы заставить время работать на вас, а не против вас. Если время — ваш союзник, у вас появляется возможность уделять лидерским качествам больше внимания — просто по той причине, что в таком случае вы сами управляете организационными вопросами, а не они направляют вашу деятельность. Когда вам кажется, что задач слишком много, а времени слишком мало, знайте: время работает против вас. Неорганизованность приводит к развитию фобий, происходящих от опасения забыть подробности многочисленных задач, которые вам предстоит решить.

Стремиться к самоорганизации мало — вы должны помогать компании совершенствовать организацию действий и процессов, которые непосредственно касаются разработки программных средств. Если вы будете действовать подобным образом, сотрудники подведомственной группы поймут, что вы способны ими руководить, — и, соответственно, будут нацелены на достижение результата. Именно это и является основной целью любой организационной схемы — сделать успех ожидаемым результатом рутинной работы. Так оно и происходит, поскольку вся деятельность сотрудников становится более эффективной, а значит, ресурсы персонала используются по максимуму.

Как превратить информацию в знания и действия

Возможно, это слишком упрощенный взгляд на повседневные обязанности менеджера, но мне кажется, что все ваши действия правомерно рассматривать как отдельные проекты по преобразованию информации в знания и действия. Некоторые из этих проектов чрезвычайно просты — к таким, к примеру, относится ответ на письмо, полученное по электронной почте. Иные, напротив, отличаются сложностью — отнесем к ним разработку плана модификации принятой в компании программной архитектуры. Во всех подобных проектах есть детали — иначе говоря, поток информации и знаний, которые требуется отслеживать, совершенствовать и каким-то образом структурировать с целью выстраивания плана последующих действий. Как правило, все действия по реализации этих ваших мелких проектов производятся за рабочим столом и за компьютером, который с ним неразрывно связан.

Теперь скажите мне: как в данный момент выглядит ваш рабочий стол? Скорее всего, он завален бумагами и папками. Это совершенно нормально и вполне допустимо, если они не лежат на одном месте неделями. Чем более запущен рабочий стол, тем больше опасность забыть о каком-то проекте, который требуется выполнить незамедлительно. Попробуйте его почистить — и, скорее всего, вы наткнетесь на информацию, которая подразумевает оперативную реакцию с вашей стороны. Наверное, вы слышали выражение «найдите для всего место и разложите все по своим местам». Это очень достойное правило; однако, следуя ему, не стоит забывать, что задача по организации «мест» является приоритетной. Сознайтесь самому себе в том, что награда «лучшей домохозяйке» за поддержание рабочего места в порядке вам не светит и стремиться к ее завоеванию бессмысленно. Что вам действительно нужно, так это создать для себя функциональные рабочие условия, не позволяющие забывать повседневные задачи, связанные с руководством группой разработчиков.

Некоторые считают прибранный рабочий стол свидетельством того, что у его хозяина не все в порядке с головой. Мне так не кажется, хотя вы имеете полное право на собственную точку зрения. Впрочем, я не могу не согласиться с тем, что если вы не способны держать свой дом в чистоте и порядке, у вас ни за что не получится руководит деятельностью других людей. Как происходит уборка дома? Вы протираете пол, пылесосите, поднимаете вещи, которые свалились на пол и мешают проходу, — иначе говоря, создаете условия, способствующие комфортному проживанию. Не кажется ли вам, что рабочее место должно быть примерно таким же? Я глубоко убежден, что удобное рабочее место является предпосылкой успешной, продуктивной работы. Вы прекрасно знаете, что неупорядоченный код существенно затрудняет процесс разработки и сопровождения программного продукта; аналогичным образом неприбранный рабочий стол доставляет руководителям немало неприятностей. Под «неприбранностью» я в данном случае имею в виду бессистемность в деле организации ваших повседневных задач, которая произрастает от неверного управления информационным потоком.

Если вы не способны держать свой дом в чистоте и порядке, у вас ни за что не получится руководить деятельностью других людей.

Бумажная волокита

Несколько десятилетий назад, прежде чем персональные компьютеры стали неотъемлемым атрибутом нашей рабочей деятельности, все детали проекта фиксировались в тетрадях и папках для документов. Сегодня — в условиях, когда основным средством структурирования выступают компьютеры, — роль бумажных носителей до известной степени снизилась, и в процессе повседневного руководства деятельностью специалистов в рамках проекта бумага уже не играет такой роли, как раньше. Тем не менее, поскольку концепция безбумажного делопроизводства далека от реализации, скорее всего, иметь дело с бумагой нам предстоит еще очень долго. Несмотря на некоторые недостатки, бумажные носители в целом неплохо справляются со структурированием информации. Если взвесить все сильные и слабые стороны бумаги, то можно утверждать (и вы, наверное, со мной согласитесь), что:

- бумажные носители очень удобны в тех случаях, когда информацию требуется изучить именно в то время и в том месте, в котором вам удобно;
- на бумаге удобно фиксировать ход дискуссии на совещании и в телефонном разговоре — писать на листке бумаги чуть проще, чем набирать текст на ноутбуке;
- бумажные носители не адаптированы для хранения, поиска, сортировки и многих других операций, с которыми блестяще справляются программные продукты;
- многие из нас воспринимают важную информацию лучше, считывая ее с бумажного носителя, и хуже, считывая ее с монитора;

- заложники собственной культуры, мы привыкли воспринимать информацию, зафиксированную на листах прямоугольной формы размером 210 × 297 мм.

Вполне возможно, что в будущем планшетные компьютеры смогут объединить в себе достоинства бумаги с широчайшими возможностями программного обеспечения. Лично я все жду, когда же появятся жидкокристаллические экраны толщиной с лист бумаги, на которых можно будет писать ручкой и карандашом, стирать записанную информацию, сохранять ее в электронном виде, а затем перезагружать на тех же листах уже другие данные. Не знаю, дождусь ли, — ведь хочется, чтобы такие экраны продавались в магазинах пачками по 500 штук и стоили не более 3,95 доллара.

В защиту бумаги отмечу, что на протяжении большей части XX века она оставалась основным носителем коммерческой информации и, скорее всего, в ближайшее время все еще будет играть важную роль. Скорее всего, вы поняли, к чему я это все говорю. Да, действительно — управляться с горами листов очень непросто. Они постоянно устаревают, вам вечно приходится думать о том, как бы не обанкротиться на одних только картриджах и т. д., и т. п.

Как же справиться с бумажной неразберихой? Придется усвоить несколько основополагающих принципов. Они излагаются на всех учебных курсах по основам бизнеса, но, по-моему, их стоит вспомнить и здесь[1]. Может быть, вам покажется, что принципы, которые я здесь излагаю, самоочевидны, но вы не представляете, какое количество менеджеров испытывают страшные мучения в попытках как-то упорядочить кипы бумаг — и все по той простой причине, что они игнорируют правила систематизации.

Итак, в том, что касается структурирования бумажных материалов, я предлагаю соблюдать нижеследующие принципы.

- Создайте стройную систему хранения документов. Для каждого проекта заведите отдельный набор папок для документов или тетрадей. В любой папке проекта нужно держать только наиболее важные бумаги и все их снабжать датой и ссылкой на источник. Папки проекта имеет смысл делить на разделы в соответствии с типами информации, как то: область действия, проектное решение, ошибки, обмен информацией, материалы совещаний и т. д.
- Следуйте системным правилам хранения документов. По завершении проекта сдайте относящиеся к нему папки в архив. Под рукой должны быть только те материалы, которые относятся к текущим проектам, а информацию о выполненных проектах следует незамедлительно перемещать в другое место.
- Каждый листок бумаги, который по тем или иным причинам появился на вашем рабочем столе, необходимо переложить в соответствующую папку. Не доводите дело до появления на столе груды бумаг. Не забывайте, что папка для входящих документов — это лишь временный резервуар и ни в коем случае не постоянное хранилище. Стоит на вашем столе появиться кипам бумаг, как вы незамедлительно начнете опаздывать с выполнением организационных функций, которые, вполне возможно, необходимы для достижения успеха.

---

[1] Я просмотрел несколько изданий по бизнесу и кое-какие из них мне понравились. В частности, возьмите на заметку издание Ronni Eisenberg. Organize Your Office (Hyperion, 1998).

- Заведите для каждого проекта несколько папок различных цветов; каждый такой цвет должен указывать на срочность действий, связанных с хранящимися в папке документами. К примеру, в красных папках можно собирать информацию, которая требует с вашей стороны незамедлительных действий. В синих папках имеет смысл хранить документы, с которыми можно ознакомиться чуть позднее, а в зеленых — все документы, связанные с денежными вопросами. Все эти правила просты и очевидны; если их не забывать, то даже после беглого просмотра документов вы сможете определиться с тем, что делать в первую очередь.
- Организуйте отдельное место для хранения литературы и соблюдайте в нем порядок. Скажем, профессиональные журналы нужно хранить отдельно от каталогов производителей. Все эти материалы следует сортировать по дате выхода и не слишком загромождать ими рабочее пространство. Если из целого журнала вашего внимания заслуживает единственная статья, вырежьте ее, положите в стопку, а от остальных страниц избавьтесь. Не стоит копить макулатуру лишь с той целью, чтобы удивить самого себя.

Скорее всего, в том, что касается систематизации бумажных документов, у вас есть какие-то собственные приоритеты и методы; возможно также, что в вашей компании существует четкая политика по этой части. Если вы не справляетесь с постоянно поступающими бумажными документами, скорее всего, это приведет к смещению приоритетов. Если бы безбумажное делопроизводство уже превратилось в реальность, то никому не нужно было бы подключать к корпоративным сетям многочисленные принтеры. Однако если уж вы ими пользуетесь, будьте любезны, разберитесь в своих распечатках.

Безбумажная волокита

Бумажные документы — это не единственная область, которую вам как менеджеру придется приводить в порядок. Несмотря на то что личные информационные секретари (Personal Information Managers, PIM) представлены на рынке в большом количестве, большинство руководителей до сих пор испытывают серьезные трудности в попытках организации электронных документов. Вполне возможно, что вы уже пользуетесь тем или иным программным продуктом для руководства проектом, он принят в вашей организации — и от него есть прок; если так, вы — счастливый человек. В то же время, по моим наблюдениям, многие компании и, в особенности, отдельные лица подходят к задаче систематизации электронных документов крайне индивидуально. Я неоднократно пытался приучить себя к применению разнообразных программ для руководства проектами и в результате понял, что все они оставляют желать лучшего. Я, пожалуй, не буду их называть. Выражусь так: все продукты одной компании, весьма популярной в северо-западных штатах Америки, оказались, с моей точки зрения, либо слишком сложными, либо чересчур ограниченными по своим возможностям, либо элементарно не соответствующими моим потребностям по части отслеживания организационных вопросов. Ну ладно, я назову имена — Microsoft Project, Team Manager (этот продукт поставляется на компакт-дисках MSDN). Не подошли мне и многочисленные видоизмененные (в плане объектной модели) версии Outlook. Lotus Notes, ACT! и многие другие личные информационные секретари вместе с программными продуктами для групповой работы также показались мне недостаточно гибкими

и не подошли для систематизации моих рабочих процессов. Я вполне допускаю, что один из этих продуктов подойдет вам. Если так случится, вы получите шанс заметно продвинуться в деле ведения стаи за собой. Если же чуда не произойдет, придется обратиться к другому методу.

Так что же делать? Ведь я программист; все продукты, которые мне не понравились, созданы программистами; кроме того, я ностальгирую по тем временам, когда писал гораздо больше кода, чем сегодня. Таким образом, ответ для меня очевиден — нужно создать собственную программу управления проектами.

Электронный администратор

Итак, я поставил перед собой задачу создать программный продукт, с помощью которого мне было бы удобно справляться со своими организационными обязанностями. Он должен был стать чем-то средним между личным информационным секретарем и полноценной программой управления проектами; естественно, я также намеревался избавиться от излишнего усложнения и ограничений, присущих обоим названным типам программных продуктов. Работать над своим проектом мне пришлось по ночам, в нерабочее время и в гордом одиночестве. За первые полгода применения своего детища я скомпоновал более 200 исполняемых версий — все это время я обнаруживал очередные ошибки и подстраивал продукт под свои потребности. Несколько раз он очень помог мне справиться с административными задачами. По крайней мере, теперь, когда он у меня есть, я четко знаю, на сколько опаздываю, мне не приходится постоянно трястись от неопределенности и сознания груза несделанных дел. У меня хотя бы есть возможность измерить этот самый груз и распечатать соответствующий отчет — из него я узнаю, сколько ночей придется провести без сна, чтобы успеть все сделать к установленному сроку.

У моего личного проекта было несколько очевидных преимуществ. Во-первых, мне не пришлось собирать коммерческие требования, поскольку сценарии вариантов действий проявлялись регулярно. Во-вторых, в роли пользователя выступал я сам, а значит, реализация потребностей пользователя ограничивалась конструированием подходящего мне интерфейса. Это ведь мечта программиста — писать программы для самого себя. На этапе бета-тестирования я также не встретил никаких особых проблем. Обнаруживая ошибки, я брал исходный код и исправлял их. Помимо прочего, при работе над этим проектом я не испытывал синдромов, часто встречающихся у нагруженных административными функциями программистов, у которых не находится достаточного времени для кодирования. Оторваться от клавиатуры ведь довольно трудно, не так ли?

Итак, далее я познакомлю вас с собственноручно разработанным программным продуктом, который, по моему скромному мнению, помогает успешно организовать производственный процесс, направленный на достижение результата. Если этот продукт вам понравится (или вы осмелитесь адаптировать его к собственным потребностям), обратитесь к приложению А. В нем приводится более подробная информация, а также сведения о том, как получить исходный код.

Задача как объект — основной организующий принцип электронного администратора

По большому счету, все задачи, которые решаются в ходе разработки программных продуктов и соответствующей административной деятельности, можно разделить на три типа.

1. *Проекты*. Все задачи аккумулируются в рамках проекта — довольно удобной классификационной конструкции, связывающей те виды деятельности, которые направлены на производство качественного кода и зарабатывание денег для компании. Отдельные проекты, впрочем, не связаны с созданием продукта, предназначенного для продажи; однако если основным приоритетом для вас является успех компании на занятых ею рынках, вы, вероятно, согласитесь со мной в том, что любые продукты воплощаются в жизнь лишь благодаря способности группы разработчиков к созданию успешного программного обеспечения.
2. *Источники*. В качестве источника — то есть субъекта, поставившего задачу, — в зависимости от ситуации может выступать человек (зачастую вы сами), процесс или комитет (члены которого обычно не в меру ворчливы). Классификация задач по источникам помогает следить за выполнением собственных обещаний — в особенности если вы имели неосторожность дать их своему начальнику.
3. *Назначенные задания*. За распределение заданий между сотрудниками рабочей группы ответственны вы. О тех людях, с которыми вам приходится работать, я говорил на протяжении трех предшествующих глав. Теперь же мы обратимся к более приятной теме неодушевленных объектов. Обычно они молчат, хотя наличие на моем компьютере синтезатора речи лишает их этой замечательной особенности. Я немного отвлекся, хотя здесь тоже прослеживается важный принцип, касающийся перевода из рядов программистов в менеджеры. Мы лучше обращаемся с «вещами», чем с людьми.

Ну ладно, хватит разглагольствовать! Лучше взгляните на рис. 4.1 — он представляет собой графическую иллюстрацию представленной выше концепции.

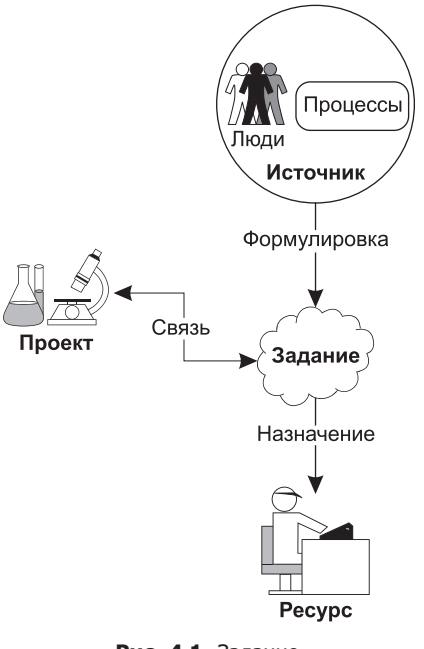


Рис. 4.1. Задание

Как и обещано, на этой схеме изображены все три отношения между задачей, с одной стороны, и информационным потоком и процессом администрирования, с другой.

За утверждением задачи — в том виде, в котором она представлена на нашей схеме — следует этап ее реализации в программном продукте. На рис. 4.2 в традициях классического двухзвенного[1] MDI-приложения с нагруженной клиентской частью (в эпоху веб-приложений[2] такая схема кажется очевидно устаревшей) изображен графический пользовательский интерфейс, реализующий представленный выше принцип.

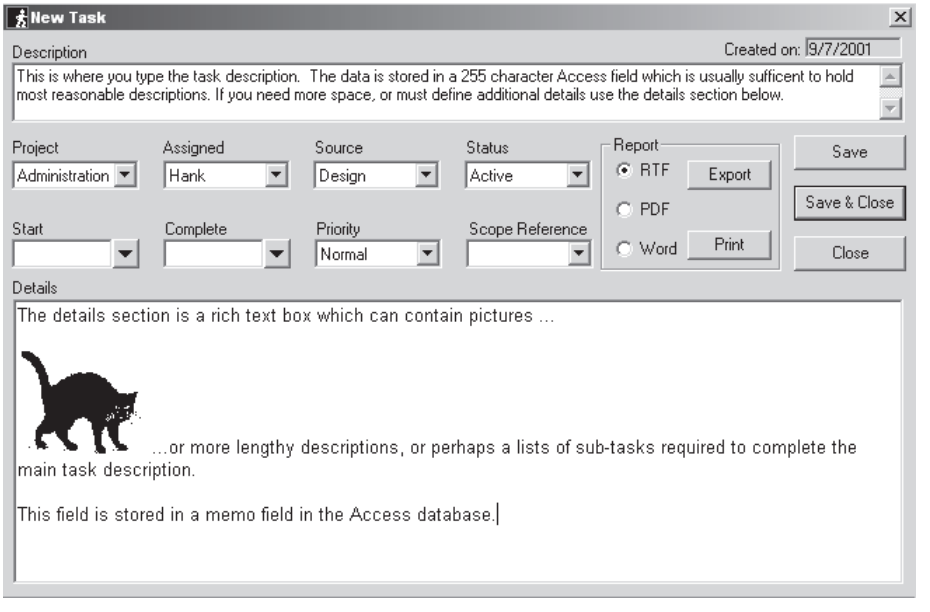


Рис. 4.2. Реализация задачи

Согласно моей задумке, в окне, показанном на рис. 4.2, должны умещаться все данные, с помощью которых можно идентифицировать задания и отследить их выполнение. Три основных отношения — источник, назначенные задания и проект — дополняются стандартными и вполне ожидаемыми параметрами: состоянием, сроками начала и завершения, а также приоритетом. Кроме того, в моей версии программы присутствует запись-ссылка на область действия[3]. В некоторых компаниях она может сослужить хорошую службу. Среди прочих нелишних характеристик стоит упомянуть возможность создания документа с данными по конкретной задаче и, естественно, возможность сохранения собранной информации в базе данных. Раздел Details я выполнил в виде форматируемого поля, в ко-

---

[1] Имеются в виду не логические, а физические звенья.

[2] Я пробовал структурировать электронного администратора в качестве веб-приложения, но такая схема не подходит моим программистам. Я их не виню; в конце концов, я должен давать задания, а они — выполнять их. Кроме того, мне не нравится ограниченность веб-интерфейса.

[3] Присутствие этой записи подразумевает наличие документации относительно области действия, в которой характеристики конструируемого продукта представлены в виде нумерованного списка.

торое можно встраивать внешнюю графику, — как известно, один рисунок способен сообщить программисту больше, чем тысяча слов.

Отображение и систематизация заданий

После того как сведения о задании зафиксированы, их, конечно, нужно отобразить — да так, чтобы из них можно было делать какие-то выводы. Как менеджер вы, естественно, вынуждены ежедневно составлять список заданий, требующих безотлагательного выполнения; кроме того, вы стремитесь к тому, чтобы каким-то образом отслеживать выполнение срочных заданий теми сотрудниками, которым вы их поручили.

Решение, к которому я обратился, предполагает наличие дочернего MDI-интерфейса, способного собирать в одном месте всю информацию о критически важных заданиях на сегодняшний день. Повторюсь: для того чтобы обеспечить соответствие конструируемого программного продукта задачам, которые он должен решать, необходимо постоянно подгонять самого себя и отслеживать работу подчиненных. В противном случае вы рискуете не справиться со своими функциями. Вспомните меткий призыв из главы 2 относительно ожидания результатов без проведения проверок, который я позаимствовал у какого-то университетского профессора. Отталкиваясь от его смыслового наполнения, я сконструировал показанный на рис. 4.3 экран.

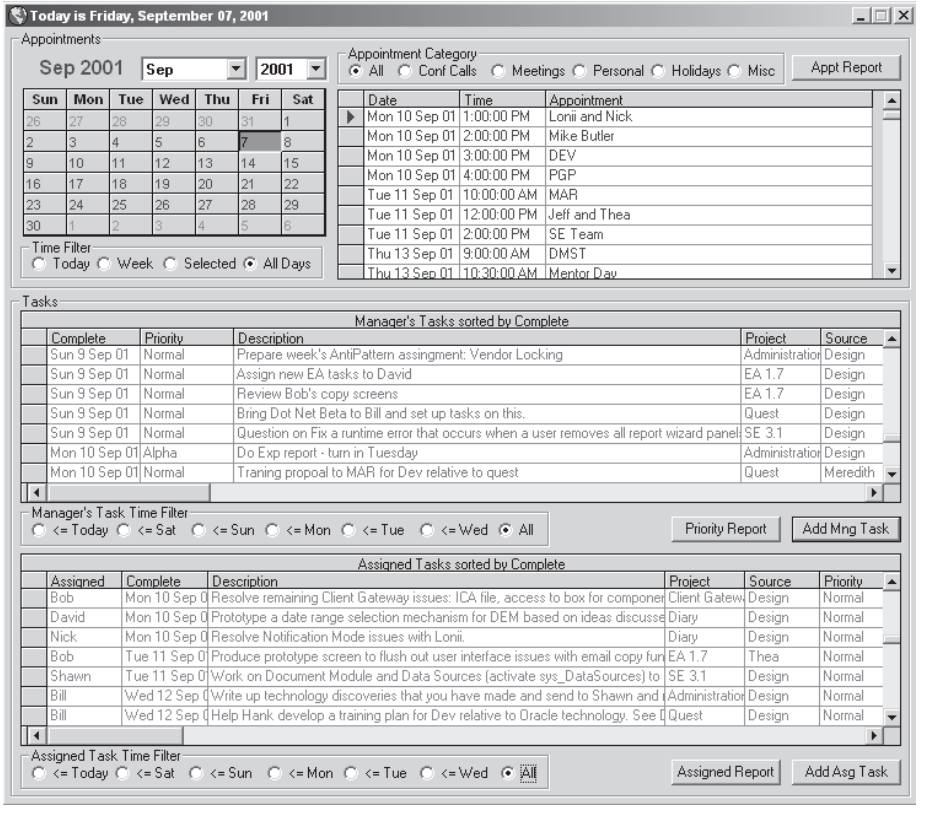


Рис. 4.3. Экран рабочего дня

Среди прочих элементов экрана рабочего дня есть календарь. Он помогает мне вспомнить, какой сегодня день, что после долгих бессонных ночей, проведенных в праведном труде, бывает довольно сложно. Кроме того, есть на этом экране и область встреч, тесно связанная со схемой задания. Строго говоря, встреча — это тоже задание, предусматривающее очное или заочное общение с людьми. Поскольку моя система включена постоянно, с помощью таймера каждую полночь представление экрана рабочего дня в ней обновляется.

В программе присутствует средство генерации отчетов о задачах и встречах руководителя, а также списках назначенных заданий. Их можно как распечатывать, так и экспортировать. Любой список заданий предусматривает возможность фильтрации по сроку завершения в виде функции текущей даты; переключатели позволяют оперативно устанавливать разные режимы: режим сегодняшних заданий и всех долгов с предыдущих дней, режим заданий до 5 дней, начиная от текущей даты, а также представление всех заданий. Каждую полночь таймер в этой форме переустанавливает заголовки фильтров, чтобы они все всегда соответствовали заданному диапазону в 5 дней от текущей даты. Для того чтобы ввести новое задание с помощью предварительно выделенного комбинированного окна Assigned, щелкните на кнопке Add Mng Task. Для назначения нового задания щелкните на кнопке Add Asg Task.

В приложении А приводятся более детальные сведения об электронном администраторе (эта программа стала моим любимым проектом), а также о примененных в ходе его конструирования методиках и компонентах.

Как выработать собственные навыки администрирования

Я уже перечислил несколько принципов организации информационного потока в пределах вашего рабочего стола. В то же время я не рекомендую слепо копировать мои методы. К ним стоит обращаться лишь в том случае, если они соответствуют конкретным организационным потребностям или имеют шанс стать хорошей отправной точкой для создания ваших собственных методик. Как сортировать бумаги по папкам, вы, вероятно, знаете, так что на этом я останавливаться не буду. Вполне возможно, что вы уже пользуетесь одним из многочисленных представленных на рынке продуктов управления проектами. Если он отвечает вашим потребностям — замечательно. С другой стороны, довольно часто недостаток гибкости применяемых инструментальных средств принуждает руководителей адаптироваться к процессам, которые не слишком согласуются с их повседневной деятельностью. По моим наблюдениям, гибкость инструментов, которые применяются для систематизации информации, вносит существенный вклад в достижение успеха. Очевидно, что возможность создания собственного программного продукта предоставляет вам максимальную гибкость по части решения организационных вопросов. Вашей задачей в этом контексте должна стать корректировка своей организационной деятельности, направленная на точное соответствие конкретным функциональным обязанностям. От этого во многом зависит способность достижения качественных результатов, причем со временем, по мере изменения условий, организационные методы нужно регулярно пересматривать.

Не следует питать иллюзий — комплексный проект с многочисленными взаимозависимостями нельзя вести исключительно средствами простых программных продуктов вроде того, что я описал в предыдущем разделе. Решение о том, подходит ли тот или иной продукт для конкретного проекта, всецело зависит от вас. Более того, вы должны сами себя оценивать по части ведения всех организационных вопросов в своем отделе. Слово «администрирование» (administration) происходит от словосочетания «добавочное министерство» (added ministry); в некотором смысле, администрирование по отношению к основным задачам не является приоритетной областью вашей деятельности. Вопрос в том, сколько сил и энергии вы затрачиваете на улаживание организационных вопросов. Необходимо всегда иметь в виду, что в процесс вашей повседневной деятельности постоянно будут вмешиваться разного рода проблемы, которые придется решать дополнительно к выполнению ваших непосредственных функций. Повышая уровень своей личностной организации, вы фактически сокращаете сроки исполнения своих первоочередных заданий.

«Непосредственные функции» — выражение весьма двусмысленное. Администрирование — это полноценная область деятельности, и заниматься ею необходимо. В то же время программисты-менеджеры воспринимают административные функции в штыки. Здесь можно провести параллель с кодированием и комментированием. Код без комментариев трудно читать. Аналогичным образом руководство без администрирования становится неорганизованным и даже (иногда) неверным по своей сути. В этой книге я исхожу из того, что вы обязаны вести сотрудников своего отдела к намеченным целям. При этом ключевую роль как в чисто внешнем, так и в фактическом аспекте лидерства играет эффективная и стройная система администрирования.

Ключевую роль как в чисто внешнем, так и в фактическом аспекте лидерства играет эффективная и стройная система администрирования

Дирижер и оркестр — еще одна подходящая аналогия. Музыкальные произведения исполняются оркестрантами, однако не будь дирижера, любая симфония превратилась бы в какофонию. Если вам доводилось наблюдать за тем, как оркестранты перед выступлением настраивают свои инструменты, вы понимаете, что такое какофония. Администратор действительно имеет много общего с дирижером. Поэтому вы должны стремиться к тому, чтобы стать почтенным маэстро. Учтите также, что в своей деятельности дирижер исходит из отличительных особенностей оркестра, которым дирижирует. Если в вашем подчинении опытные и профессиональные сотрудники, административным вопросам не понадобится уделять много времени. Если же вам приходится руководить новичками, то вы вряд ли обойдетесь без формальной организационной структуры.

Как организовать контроль

«Контролировать или не контролировать?» — Гамлет тоже пытался решить подобную дилемму. Правда, в его случае онтологические последствия решения оказались

куда более серьезными. Ваши мучения обещают оказаться не столь жизненно важными. В то же время имейте в виду, что достижение результата в компании зависит от самых разных процессов. Некоторые из них поддаются контролю с вашей стороны, другие — нет. Именно поэтому важно, как вы организуете процессы, как справляетесь с теми процессами, которые можно контролировать. Личностная организация позволяет справляться с внешними дезорганизующими факторами, какими бы серьезными они ни казались. Будьте уверены — остальные сотрудники компании отчаянно борются с организационными хитросплетениями так же, как и вы. Присмотритесь к ним. Перенимайте у них удачные идеи и не забывайте предлагать им свои.

Рассмотрим список подконтрольных и неподконтрольных областей (табл. 4.1).

Таблица 4.1. Подконтрольные и неподконтрольные области

Подконтрольные области	Неподконтрольные области
Методы и действия, направленные на упорядочение и систематизацию информации	Потребность окружающих людей делиться с вами информацией
Распределение конкретных задач между сотрудниками отдела	Выбор проектов, над которыми вам предстоит работать. Проекты могут нравиться или не нравиться, но работать нужно со всеми
Архитектура программного обеспечения	Коммерческие задачи, решение которых ложится на ваши плечи
Время, фактически выделяемое на выполнение заданий	Время, которое вы должны уделить решению задач в соответствии с внешними ожиданиями
Ваши ожидания относительно собственной продуктивности	Ожидания окружающих относительно вашей продуктивности
Ваше отношение к трудным с точки зрения общения людям	Трудные в общении люди, непосредственно вам не подчиняющиеся

Этот список можно расширять очень долго и довести до нескольких страниц. Мне хочется, чтобы вы осознали важность проблем, связанных с контролем, и поняли, что самоорганизация, направленная на достижение результата, заключается в достижении максимального контроля над рабочей ситуацией. Стоит немного углубиться в проблемы контроля, и вы поймете, почему так происходит.

Информационный поток

Все мы живем в информационную эпоху, в которую для успешного производства программного обеспечения необходима способность концентрироваться на решении той или иной коммерческой задачи. Процесс этот предполагает анализ информационного потока, проходящего через компанию и относящегося к определенным коммерческим потребностям, и усвоение этой информации. Ответить на все вопросы сразу и самостоятельно никогда не удается — следовательно, отдельные проблемы приходится решать с помощью других людей. Иногда объем данных, с которыми мы регулярно сталкиваемся, буквально сводит с ума. Впрочем, не зря в предыдущих разделах я говорил об организации деталей проекта — эти действия способны не допустить ступор, наступающий при организационной перегрузке.

Как получилось, что традиционные мифы и суеверия эволюционировали до состояния современной науки? Дело в том, что со временем к изучению природы во все большем объеме стали привлекаться научные методы. Поскольку природа в высшей степени информативна, первым этапом любого исследования является классификация. Отсюда проводим прямую параллель с организационными методиками. Вы имеете полное право не понимать какую-то часть тех данных, с которыми сталкиваетесь; однако если вы сумеете классифицировать их, разбить на отдельные тематические группы, считайте, что первый шаг по направлению к знаниям сделан.

Объем воспринимаемой информации часто не поддается контролю, однако методы, применяемые для ее систематизации, вполне управляемы. Отчасти работа руководителя напоминает труд библиотекаря; хочу, впрочем, заметить, что применительно к хорошему руководителю эта аналогия носит исключительно позитивный характер. Зная, куда обратиться в поисках необходимой информации, вы получаете неоспоримое преимущество перед теми руководителями, которые подходят к своим обязанностям неорганизованно и потому не ориентируются в информационном пространстве.

Столкнувшись с новым информационным блоком проекта, задайте себе несколько вопросов.

- Как новые данные отражаются на области действия проекта, на проектном решении, на конечном сроке сдачи проекта?
- Надежен ли источник информации; если нет, то можно ли его проигнорировать?
- Следует ли реагировать на поступление информации незамедлительно или можно немного подождать?
- Где и как сохранить информацию с тем, чтобы при необходимости к ней можно было бы оперативно обратиться?
- Каков срок действия информации? Когда она становится неактуальной?
- Как данная информация соотносится с тем, что уже известно о проекте?

Эти и подобные вопросы направлены на систематизацию данных. Систематизируя данные, вы обретаете контроль над ними, и, соответственно, они теряют контроль над вами.

Назначение заданий

Правда, было бы замечательно, если бы мы могли работать только над теми проектами, которые нам лично интересны? К сожалению, в долгосрочной перспективе такой подход нежизнеспособен. Иногда задачи, которые ставит перед нами наше дело, идут вразрез с нашей эмоциональной реакцией на них. Таким образом, тщательное распределение заданий есть наилучший метод поддержания заинтересованности в их исполнении со стороны ваших сотрудников (да и с вашей стороны тоже). Люди бывают разные: одни лучше приспособлены к одним проектам, другие — к другим. В главе 1 эта проблема обсуждалась в контексте пород программистов.

Следует, правда, учитывать, что ваши возможности по наделению всех сотрудников интересной работой ограничены. Если вы хотите остаться на работе,

придется выполнить все проекты, отведенные вашему отделу. В частности, вам не избежать необходимости разработки проблем, которые вам совершенно безразличны. Как преодолеть упаднические настроения, которые часто сопровождают начало очередного нудного проекта по сопровождению кода? Попробуйте перераспределять задания. В результате вы сможете избежать ситуации, при которой все самые утомительные функции ложатся на одного человека. Эти вопросы находятся в зоне вашего контроля. Варьируйте задания, перераспределяйте их между сотрудниками ежедневно или еженедельно — для этого у вас есть все возможности. Многие усматривают в разнообразии смысл жизни; по моему мнению, программистам разнообразие совершенно необходимо — оно помогает выводить наши мозги «на пик формы».

Не зная индивидуальных пристрастий сотрудников, вы не сможете обеспечить их заинтересованность. Помните, что качественный код пишут заинтересованные программисты. Впрочем, у дисциплинированных программистов код получается еще лучше. Сделать программистов счастливыми не всегда в ваших силах (в конце концов, это не ваша работа), однако сформировать из своих сотрудников дисциплинированную группу, готовую к работе над трудными проектами, вы можете. Для этого необходимо поддерживать разнообразие функций, которые они регулярно выполняют.

Архитектура

Поскольку вы являетесь техническим руководителем процесса разработки программных продуктов, очертания архитектуры системы в значительной степени зависят от вашей воли. Задачи, решаемые в контексте архитектуры, не всегда поддаются контролю, но, в конце концов, в нашем деле это вполне предсказуемая ситуация. Как я уже говорил, функции менеджера не всегда доставляют удовольствие. Реализовать себя помогает способность получить удачное решение с помощью имеющихся технических навыков и вывести свою компанию, исходя из ее коммерческих потребностей, на достойное положение в занимаемом ею сегменте рынка.

Роль технического лидера подробно рассматривается в главе 6. Вы еще успеете ее прочитать, а пока запомните, что контроль над программной архитектурой, применяемой в компании, в значительной степени определяет успешность конечного результата вашей деятельности. Какое отношение, спросите вы, имеют организационные навыки к архитектуре? Непосредственное, отвечу я. Методики, применяемые при систематизации потока административной информации, полностью применимы к проектированию программных средств. Подумайте, как классифицируются программные требования в расчете на их реализацию в объектах кода. Не кажется ли вам, что инкапсуляция кода во многих отношениях подобна наполнению папки для документов проекта? Я считаю, что совершенствование организационных навыков в одной области помогает добиться лучших результатов в другой.

Традиционно считается, что левое полушарие у программистов развито лучше, чем правое, и что они соответственно ориентированы на логическое и аналитическое мышление. Естественно, без мозговой деятельности в нашей работе не обойтись. Организационные способности родственны логике программирования.

Так что не стоит думать, что структурирование рабочих функций — это деятельность, не предполагающая приложения мыслительных усилий. Аналогичным образом для создания качественной программной архитектуры требуются организационные навыки и умение мыслить абстрактно.

Рабочие часы

На продолжительность вашего рабочего дня влияет великое множество факторов. Скорее всего, вы уже свыклись с тем, что сверхурочные (помимо стандартных 40 часов в неделю) в вашем положении есть неизбежное зло. Возможно, это действительно так. Когда компьютерная революция только начиналась, все мы надеялись на то, что она приведет к повышению производительности труда, что, в свою очередь, облегчит планирование продолжительности рабочей недели. Революция подошла к концу, но результаты в этом отношении двойственны. С одной стороны, с помощью компьютеров мы действительно сумели повысить эффективность своего труда. Часто это означает, что более серьезные результаты достигаются за меньшее время. С другой стороны, исходя из собственного опыта, я могу заключить: возможность выполнения большего числа заданий приводит к тому, что мы начинаем больше работать, в результате рабочая неделя неизбежно удлиняется, и вести нормальную жизнь вне рабочего пространства становится все труднее. Если вы будете работать дольше, чем обычно, вашему примеру обязательно последуют подчиненные. Эффект подобного рода изменений неоднозначен. Если вы человек несемейный, то, скорее всего, продление рабочего времени никак не скажется на вашей личной жизни. Те же ваши сотрудники, у которых есть семьи (а им, как известно, неплохо бы уделять определенное внимание), вряд ли будут радоваться удлинению рабочей недели. При принятии решения о том, какой пример подавать подчиненным, этот фактор нужно обязательно учитывать. В отрасли производства программного обеспечения переработка встречается сплошь и рядом. В то же время вы должны учитывать, что продолжительная беспрерывная рабочая деятельность неизбежно ведет к снижению производительности.

Постарайтесь находить баланс между тенденцией к переработке и щадящим распределением времени. Повышение дисциплины устраняет необходимость в переработке, и, таким образом, к удлинению рабочего дня приходится прибегать все реже. Именно в этом состоит одна из многочисленных задач высокоорганизованной административной деятельности — повысить эффективность работы в стандартные рабочие часы и тем самым сократить переработку.

Естественно, я отдаю себе отчет в том, что современная производственная действительность отнюдь не способствует сохранению стандартной 40-часовой рабочей недели. Но это совершенно не означает, что у вас нет ресурсов для сопротивления этой тенденции. Уверяю вас: люди, лежащие на смертном одре, редко сожалеют о том, что проводили в офисе слишком мало времени. При планировании переработки это обстоятельство нужно обязательно иметь в виду. Кроме того, чутко следите за признаками изнеможения — как своих сотрудников, так и самого себя. Учтите, что восстановление после года напряженной работы проходит значительно болезненнее и занимает значительно больше времени, чем усилия по самоорганизации в роли менеджера.

Восстановление после года напряженной работы проходит значительно болезненнее и занимает значительно больше времени, чем усилия по самоорганизации в роли менеджера.

Ожидания

Не сомневаюсь, что вы предъявляете к себе самые что ни на есть высокие требования, — в противном случае вы вряд ли стали бы руководителем. Все это замечательно, ставить перед собой масштабные цели очень полезно. В то же время вы просто обязаны сопоставлять свои амбиции с реальностью. Реалист четко осознает, что время от времени он может действовать не лучшим образом. Реалист также понимает, что контролировать ожидания окружающих относительно него самого невозможно — если только он не сторонник раздачи невыполнимых обещаний. Кстати, об обещаниях — с ними нужно соблюдать осторожность. Именно от них зависит, какие требования к вам будут предъявлять ваши сотрудники и ваши коллеги. Держать чужие ожидания относительно себя самого под контролем вы сможете лишь в том случае, если будете благоразумны в своих обещаниях. Ричард Карлсон (Richard Carlson), заслуживший за свою книгу «Don't Sweat the Small Stuff» исключительной похвалы, рассуждает об обещаниях следующим образом:

> «Некоторые обещания, которые мы даем окружающим людям, на первый взгляд, и не кажутся таковыми. Иногда мы даем их, сами того не сознавая. Чего стоят выражения типа «я тебе перезвоню позже», «я подъеду к тебе на работу», «на следующей неделе я вышлю тебе экземпляр моей книги», «я с удовольствием куплю ее тебе», «если тебя понадобится сменить, дай мне знать». Даже невинные в первом приближении фразы типа «без проблем» представляют для сказавшего некоторую опасность — дело в том, что их часто воспринимают как предложение выполнить некоторое действие, хотя на самом деле вы либо не хотите этим заниматься, либо не располагаете для этого достаточными ресурсами. Фактически, сказав так, вы позволили собеседнику высказать к вам более серьезную просьбу, чем раньше, — ведь это не проблема!»[1]

Отдельные личности склонны ожидать от руководителей больше, чем те способны сделать. С такими ожиданиями нужно смириться. Некоторые начальники — я имею в виду тех, кто находится выше вас в руководящей иерархии — предъявляют к своим подчиненным чрезмерно высокие требования, но при этом не удосуживаются посвящать их в подробности своих замыслов. Если это именно ваш случай, не вините себя. На самом деле виноват ваш начальник, а читать его мысли вы пока не научились, так? О том, как взаимодействовать с начальником, мы поговорим более подробно в главе 9. Пока что запомните: дать отпор несбыточным ожиданиям можно за счет ваших организационных способностей. Обязательно уточняйте, что от вас хотят. Если ваш начальник темнит относительно того, чего он от вас ожидает, поднажмите на него, заставьте четко сформулировать требования. Без четко определенных исходных данных добиться успеха в области конст-

[1] Richard Carlson, Don't Sweat the Small Stuff at Work (New York: Hyperion, 1998), p. 74.

руирования программных средств невозможно. Организованность в этом процессе есть лучший способ избежать ситуаций, при которых первая версия продукта оказывается для остальных подразделений компании неожиданным и нежелательным сюрпризом.

Взаимоотношения

Возможно, вы слышали такое высказывание: «контролировать проблему трудно, но контролировать отношение к проблеме в ваших силах». Для того чтобы осознать справедливость этой мысли, мне потребовалось довольно много времени. А как думаете вы? Если ваше отношение к этому принципу скептическое, скорее всего, во взаимоотношениях с трудными людьми вам суждено проигрывать. Организованность отнюдь не ограничивается умением раскладывать документы по папкам и вводить данные в программу управления проектом. Куда важнее уметь должным образом обращаться с трудными подчиненными. Ларри Константайн (Larry Constantine) — ведущий исследователь проблем персонала в проектах по разработке программных средств — мыслит в этом контексте следующим образом:

> «Попытки справиться с трудными ситуациями следует начинать с постановки ряда вопросов. По моему опыту, задав нужный вопрос, вы кардинально повышаете шансы на получение нужного ответа. Вместо того чтобы возмущаться, почему с тем или иным сотрудником так сложно говорить, я предпочитаю задаться вопросом о том, почему у меня возникают трудности во взаимодействии с ним. Я прекрасно понимаю, что вылавливать мелкие недочеты в деятельности коллег значительно проще, чем устранять собственные недостатки, — даже если они очень существенны. И все же я утверждаю, что любая нервотрепка в процессе общения с трудным человеком — это одновременно возможность лучше узнать самого себя. Придерживаясь этого принципа, вы, скорее всего, со временем обнаружите, что людей, с которыми вам трудно общаться, остается все меньше»[1].

Какими методами бороться с трудностями во взаимодействии с персоналом? Мне кажется, метод, предложенный в предыдущей цитате, полностью себя оправдывает. В вашей административной деятельности он должен занять не менее значимое место, чем умение приводить в порядок свой рабочий стол. Поскольку самые трудноразрешимые и далеко идущие проблемы, которые приводят к дезорганизации, порождаются самими людьми, конструктивный подход к решению проблем персонала представляется мне основной предпосылкой успешной организаторской деятельности.

Поскольку самые трудноразрешимые и далеко идущие проблемы, которые приводят к дезорганизации, порождаются самими людьми, конструктивный подход к решению проблем персонала представляется мне основной предпосылкой успешной организаторской деятельности.

[1] Larry L. Constantine, Beyond Chaos: The Expert Edge in Managing Software Development (New York: Addison-Wesley, 2001), p. 4.

Как повысить организованность в масштабах всей компании

Вплоть до этого момента мы говорили об организационных навыках индивидуального характера, основываясь при этом на принципе, согласно которому чем лучше человек организован, тем более эффективно он выполняет свои руководящие обязанности. В масштабе организации как целого этот принцип играет даже большую роль. В большинстве случаев вы вместе со своими подчиненными входите в состав более крупной организационной структуры; по этой причине степень вашего влияния на деятельность компании обусловливается тем, насколько успешно вы реализуете общие для компании цели. В том случае, если программное обеспечение является основным продуктом компании, руководимый вами отдел может играть одну из двух взаимоисключающих ролей: узкого места или источника постоянного корпоративного успеха. Решение о том, какая роль предпочтительнее, принимаете вы, не забывая при этом, что ваша деятельность требует взаимодействия с другими менеджерами компании.

Какова основная причина, по которой ваш отдел выделен в самостоятельную структурную единицу? При выборе организационной схемы, направленной на достижение поставленных целей, вы должны исходить из ответа на этот вопрос. Рассмотрим несколько возможных мотивов к выделению вашей рабочей группы и выявим варианты логического воздействия этих факторов на организацию программных процессов и процедур. Возможные мотивы в расчете на их дальнейший анализ приведены в табл. 4.2.

Таблица 4.2. Мотивы, определившие необходимость выделения вашего отдела в автономное структурное подразделение компании

Мотив	Воздействие
Производить гениальные продукты и тем самым зарабатывать для компании деньги	Путем продуктивного и эффективного использования компетентных специалистов и прочих ресурсов свести непроизводительные издержки к минимуму
Забавляться с технологией	Оптом скупать свежие программные средства и бесцельно генерировать забавные программы
Пройти очередной этап к реализации ваших непомерных амбиций	Работать без отдыха в надежде на повышение или, пользуясь своим положением, участвовать в интригах и добиваться повышения во что бы то ни стало любыми средствами

Лишь первый мотив из трех вышеперечисленных можно признать допустимым. Остальные либо вторичны, либо в корне неверны. О существе третьего мотива я расскажу в главе 7, посвященной обратной стороне лидерства. Вы должны четко уяснить, что амбиции вполне допустимы, но лишь в том случае, если на первое место вы ставите планы компании и лишь на второе — свои планы. Второй мотив, приведенный в табл. 4.2 (забавы), при удачном стечении обстоятельств становится побочным продуктом первого.

Чтобы направить вас в верном направлении в смысле совершенствования организации в масштабах всей компании, необходимо более детально ознакомиться с первым мотивом. Организованность прежде всего проявляется в наиболее ви-

димой, осязаемой области вашей деятельности — а именно в процессе создания и руководства разработкой программного обеспечения. Обрести полный контроль над этой сферой вам вряд ли удастся, однако в том, что в планировании и осуществлении соответствующих операций вы исполняете ключевую роль, сомневаться не приходится.

Мою книгу трудно отнести в одну категорию с формализованными учебниками по руководству проектами. Но факт остается фактом — для того чтобы преуспеть в области лидерства, вы должны стать хорошим руководителем проекта. В литературе, имеющейся на рынке, эта область описана очень комплексно, и ряд рекомендуемых работ по данной теме есть в разделе «Библиография». Шаг за шагом вам предстоит овладевать все новыми и новыми навыками руководства проектами и в особенности — специальными методиками управления проектами по разработке программного обеспечения. По словам автора одной из моих самых любимых книг на эту тему: «основная причина неудач в процессе разработки программных продуктов заключается в неадекватном руководстве проектами»[1]. Обратите внимание на ключевое слово — «адекватность». Действительно, наличие организационных навыков, ориентированных на управление проектами, есть основной залог успешной реализации последних. На первый взгляд приведенная цитата кажется чрезмерно упрощенной. В то же время, если вы ознакомитесь с рекомендуемыми мною литературой и предложениями по руководству проектами, многочисленные причины, по которым это высказывание представляется справедливым, станут для вас очевидными.

Руководство продуктами

Вы заняты в процессе разработки программного продукта (или программной услуги), необходимость в котором обусловливается коммерческими потребностями компании. Вы ответственны за разработку; в то же время есть специалисты, которые занимаются определением, специфицированием, маркетингом и многими другими аспектами создания программных продуктов. Нет сомнений, что в процессе специфицирования продукта вы принимаете непосредственное участие, но все же ваша основная обязанность заключается в том, чтобы его создать. Тем сотрудникам, которые больше склонны к работе с клиентами и распутыванию хитросплетений бизнеса, поручают общие полномочия по координации производства, включая принятие решений по содержанию последующих версий, по расстановке приоритетов, по исправлению найденных в программе ошибок, по совершенствованию или корректировке механизмов взаимодействия пользователя с продуктом. Конечно, во всех этих областях у вас есть собственное мнение, и, скорее всего, вы можете оказаться полезным людям, которые занимаются ими вплотную. И все же координаторами и инициаторами всех этих процессов должны выступать другие.

На протяжении нескольких десятилетий существовал определенный тип программистов, представители которого предпочитали делать собственными силами все — от формулирования концепции продукта до его выпуска. В условиях современных американских корпораций такие индивиды встречаются крайне редко.

---

[1] William H. Brown et al, AntiPatterns in Project Management (New York: John Wiley & Sons, 2000), p. xxi.

К величайшему сожалению некоторых разработчиков (в основном тех, которых я отношу к породе ковбоев), программы больше не пишутся в гаражах и спальнях. То время, когда сногсшибательный программный продукт могли создать двое или даже один программист, ушло в историю. В современном мире программное обеспечение из категории занимательных новинок перешло в категорию продуктов первой необходимости.

То время, когда сногсшибательный программный продукт могли создать двое или даже один программист, ушло в историю. В современном мире программное обеспечение из категории занимательных новинок перешло в категорию продуктов первой необходимости.

Если вы работаете в небольшой компании, скорее всего, выделение в ней отдела руководства продуктами будет связано с большими трудностями или просто окажется слишком дорогим и неэффективным. Это, впрочем, не отменяет необходимости в выполнении функций таких отделов. Возможно, они даже войдут в ваши обязанности. В таком случае мои вам соболезнования — мне приходилось заниматься обоими видами деятельности, и, скажу я вам, переключаться между двумя ролями очень трудно. В идеале вам следует сориентировать своих подчиненных не на описание продуктов, а на их разработку. Я совершенно не хочу сказать, что программисты не способны к восприятию коммерческих потребностей, — напротив, чем больше осведомлен программист, тем лучше. Тем не менее нужно иметь в виду, что программист призван реализовать существующее описание; переделывать его заново совершенно не стоит. По моим наблюдениям, даже без участия сотрудников из других отделов программисты способны мгновенно организовать разрастание области действия. Продукт легче всего разработать, если он четко описан. Более того, им удобнее управлять — по крайней мере, по сравнению с не в меру инициативными кодировщиками, бредящими очередной специальной функцией.

Насколько тесно вы должны взаимодействовать с группой руководства продуктами? В общем-то довольно тесно. Вы или один из ваших ведущих сотрудников должны сверять свои действия со специалистами по коммерческим аспектам на каждом этапе описания продукта. Мало того, что эта схема способствует более полной передаче знаний между двумя группами; следуя ей, вы обеспечиваете более высокое качество специфицирования. Связано это с тем, что с самого начала процесса все неоправданные или нереализуемые характеристики из проекта исключаются. Сложно представить себе более бессмысленную ситуацию, чем та, при которой срок завершения работы над проектом и коммерческие требования просто передаются из одной комнаты в другую. Классический процесс следует направлять таким образом, чтобы все сотрудники компании в своей деятельности руководствовались едиными коммерческими задачами и решениями.

Классический процесс следует направлять таким образом, чтобы все сотрудники компании в своей деятельности руководствовались едиными коммерческими задачами и решениями.

Определение проекта

Обратите внимание: речь идет не о «руководстве проектом». Представить себе неопределенный проект довольно сложно. Процесс определения проекта следует непосредственно за описанием продукта. В этом контексте вам придется приложить свои административные способности. В предыдущей главе, как вы помните, мы говорили о том, чем нереалистичный цикл разработки продукта отличается от реалистичного. В основном различия между проектом, организованным по принципу Алисы в Стране Чудес, и проектом, который имеет шанс увенчаться созданием качественного продукта, кроются в деталях. За этапом определения следуют этапы специфицирования, проектирования, макетирования и многие другие итеративные процессы, формирующие очертания процесса в целом. Все эти процессы можно отнести к разработке, однако следует иметь в виду, что цикл разработки выводится исходя из контекста различных мелких элементов проекта.

При организации проекта вы должны отталкиваться от опыта работы с предыдущими проектами — вне зависимости от того, насколько они были успешными или, наоборот, провальными. Наибольший воспитательный эффект мы получаем от неудач (хотя и успехи в этом отношении очень полезны). Прежде чем подписаться на методику программной инженерии, основательно подумайте. Методика эта полностью применима к некоторой части крупных проектов, однако в том, что касается программного обеспечения, мастерство оказывается важнее инженерии. Это мое мнение. Его разделяют многие другие специалисты, но как к нему относиться — решайте сами. В конце концов, купив эту книгу, вы заплатили за мое мнение определенную цену. Выражусь иначе: если ваша группа сможет договориться со спонсорами по поводу этапов работы над проектом, то при утверждении сроков завершения работы вряд ли вам придется тыкать пальцем в небо. Дата выпуска программных продуктов часто рассматривается как движущаяся цель; чтобы справиться с неопределенностью, постарайтесь уяснить одну простую вещь — выпуск невозможен без предварительного комплексного тестирования. Очевидно, что тестирование проводится после кодирования, кодирование после проектирования и т. д. Я совершенно не хочу навязать вам какой бы то ни было процедурный шаблон с условием обязательного применения во всех проектах. Я просто пытаюсь напомнить о том, что прежде конструирования нужно тщательно определить и структурировать процессы, реализуемые в рамках проекта.

Если к руководству группой программистов вы приступили совсем недавно, читайте как можно больше литературы (а именно этим вы сейчас занимаетесь) и регулярно советуйтесь с начальником. Наличия технических навыков еще недостаточно для разработки проекта создания программного обеспечения. Эта деятельность требует некоторого опыта работы на руководящих должностях и коммерческого благоразумия. Помните, что ваши ресурсы ограничены; соответственно привыкайте к тесному сотрудничеству с другими подразделениями вашей компании.

Руководство процессами

Как известно, изменения в нашей жизни есть единственное постоянное явление. Этот принцип вы слышали не раз, не так ли? Даже в курсе по основам физики

утверждается, что в замкнутых системах следует ожидать повышения энтропии. Так почему же тогда мы пишем книги, читаем код, занимаемся другими структурированными видами деятельности? Надо полагать, что-то (или Кто-то) способствует упорядочению, которое противостоит естественной, присущей всей Вселенной тенденции к дезорганизации. «Что это он такое мелет? — спросите вы. — Уж не хочет ли он сказать, что мне предстоит стать "Богом управления процессами"?» Да, именно это я и имею в виду. Хотите более замысловатый титул — можете называть себя «владыкой изменений». Главное, как вы понимаете, — не наименование, а действие. В рамках общей, единой для вашей компании стратегии вы должны руководить процессом определения продуктов и проекта.

Изменения делятся на два основных типа: намеренные и случайные. Изменения, относящиеся к первому типу, обычно планируются; вторые же, как правило, непредсказуемы. Если вы ориентированы на успех, старайтесь тщательно управляться с намеренными изменениями — тогда вы сможете получить определенный контроль над случайными изменениями. Подумайте, какое воздействие модификация продукта окажет на сетевую инфраструктуру? А как насчет изменения механизма взаимодействия с пользователем во второй версии продукта? Учитываете ли вы все эти моменты? Кодирование не есть изолированный процесс. По словам Томаса Мертона (Thomas Merton), «нас согревает огонь, а не дым; через океан мы плаваем на кораблях, а не на волнах, которые они оставляют за собой»[1]. К сожалению, слишком часто программные продукты создаются без достаточного анализа воздействия технических решений на окружающую среду — «дым» и «волна» наших усилий в таких случаях приводят к дестабилизации деятельности компании.

Деятельность по организации изменений нужно вести на всех без исключения этапах разработки проекта — от зарождения замысла до завершения.

Выражаясь более приземленно, деятельность по организации изменений нужно вести на всех этапах разработки — от зарождения замысла до завершения. Если никто не позаботился о том, чтобы построить хотя бы общий шаблон для получения информации о воздействии ожидаемого (или неожиданного) изменения, в процессе разработки вас ждут серьезные трудности. Чтобы выявить подводные камни изменений, достаточно задать ряд простых вопросов. Имейте в виду: задавать их следует применительно к любой предполагаемой модификации продукта.

- Каким представляется воздействие изменения на архитектуру системы и процесс сопровождения?
- Как предполагаемое изменение воздействует на сетевую инфраструктуру, в которой оно будет проводиться?
- Как предполагаемое изменение повлияет на способность пользователя эффективно и продуктивно взаимодействовать с данным программным продуктом?
- Какое влияние предполагаемое изменение окажет на действия сотрудников отделов, которым его придется испытать на собственной шкуре?

---

[1] Thomas Merton, No Man Is an Island (New York: Harcourt Brace Jovanovich, 1955), p. 117.

Получив ответы на все эти вопросы, считайте, что вам удалось наладить процесс организации изменений, и исходя из результатов в рамках разработки программного продукта можно будет уже принимать те или иные решения. Мне кажется, что совещания с руководителями других отделов, выступающими в роли заинтересованных в успехе компании лиц, по вопросу об организации изменений следует проводить еженедельно. Координируя изменения систематически, вы обретаете контроль над дальнейшей судьбой разрабатываемых продуктов и культивируете ресурсы их поддержки. Чем тушить пожары, их лучше предотвращать, не так ли? Согласно этому принципу наличие стройной политики организации изменений позволит вам выстроить стратегические планы и отказаться от практики еженедельного созыва экстренных совещаний.

Тестирование

Не сомневаюсь, что вы слышали знаменитый слоган Java: «что написано однажды, исполняется везде». На самом деле, «что написано однажды, придется тестировать везде». Это правило применимо ко всем без исключения языкам. Не стоит доверять завершающий этап тестирования исключительно сотрудникам своей группы. Если в вашей компании нет специальной группы тестирования, организуйте ее. В крайнем случае сделайте так, чтобы сотрудники проверили код друг друга. Старайтесь, чтобы в ходе цикла разработки программисты-новички занимались тестированием не меньше, чем кодированием. Это позволит им перенять ценный опыт своих более квалифицированных собратьев.

Если программист считает, что справился со своим кодом «неплохо», насторожитесь! Дело в том, что эпитет «неплохо» можно трактовать по-разному. В любом случае в деле выявления ошибок контрольный сценарий, написанный бизнес-экспертом, приносит большую пользу, чем программист, просматривающий функцию собственного сочинения. Ни один человек не способен в одиночку адекватно оценить последствия побочных изменений в программном продукте. Наличие группы тестирования, сотрудники которой также ответственны за развертывание, есть необходимое условие достижения успеха. Тестеры должны быть вашими лучшими друзьями. Именно они помогают выпускать качественные продукты; они же составляют первую линию защиты от некачественных графических пользовательских интерфейсов.

Руководство инфраструктурой

Многие считают, что физические условия, в которых ежедневно трудятся сотрудники компании, не слишком существенны. Если и вы придерживаетесь этого мнения, подумайте еще раз. В главе 3, рассуждая о том, как разбираться с внешними раздражителями, я мельком упомянул проблему рабочего пространства. Стандартные офисные клетушки не подходят для программистов. Совместное пользование одним компьютером — это тоже не про нас. Не менее разрушительной в контексте деятельности программистов представляется практика непрекращающихся боевых действий с системными инженерами с целью получения доступа к важным системным ресурсам. Все эти элементы физического окружения, которые совершенно необходимы для успешной деятельности группы программистов, стоят денег, и немалых. Задарма ничего не выйдет. С другой стороны, если сопоставить

продуктивность (в категориях времени и стоимости) работы в плохих условиях с продуктивностью в нормальной рабочей среде, мы придем к выводу, что вложения в физическую инфраструктуру вполне оправданы.

Организуйте сотрудничество и уединение

На то, чтобы говорить, и на то, чтобы думать, у программистов должно быть время. Некоторые утверждают, что тем и другим они могут заниматься одновременно; впрочем, исходя из результатов многолетних наблюдений, я с подозрением отношусь к подобного рода заявлениям. По моему мнению, для того чтобы программисты достигали в своей деятельности определенных успехов, у них должна быть возможность, с одной стороны, некоторое время поработать в коллективе, с другой — пораскинуть мозгами в полном одиночестве. У каждого программиста должно быть собственное помещение с дверью, которая действительно закрывается. Конкретный метраж обусловливается теми средствами, которые вы готовы вложить в улучшение физических рабочих условий. Скажу лишь, что если на каждого программиста будет приходиться менее 130 квадратных футов, вы рискуете нарваться на неприятности. Как мне удалось вычислить эту величину с такой точностью? Дело в том, что 130 квадратных футов — это средний метраж спальни американского подростка. Если уж подростки, проводящие свои юные годы на таком пространстве, способны добиваться определенных успехов, то же можно сказать о программистах.

Для того чтобы программисты достигали в своей деятельности серьезных успехов, у них должна быть возможность, с одной стороны, некоторое время поработать в коллективе, с другой — пораскинуть мозгами в полном одиночестве.

Поскольку в ваши обязанности входят регулярные встречи с сотрудниками в формате «один на один», пространство вашего кабинета следует немного увеличить по сравнению со средним показателем. Не обольщайтесь — роль руководителя сама по себе не дает вам права на большую площадь; она нужна лишь для того, чтобы вы более эффективно выполняли свои функции. Кроме того, для проведения полноценных собраний с участием всех сотрудников отдела вам нужен конференц-зал, оснащенный электронным табло. Все мы наслушались историй о том, как легенды нашей высокотехнологичной отрасли регулярно режутся друг с другом в видеоигры. Вот и вам, если, конечно, позволят ресурсы, желательно организовать отдельное пространство для отдыха сотрудников. Если же средства не позволяют завести такое помещение исключительно для программистов вашего отдела, постарайтесь решить эту проблему совместно с другими подразделениями.

Удаленная работа, о которой мы также упоминали в предыдущей главе, — это хороший способ свести к минимуму влияние внешних раздражителей и решить проблему уединения; в то же время с точки зрения сотрудничества такая практика дает не слишком хорошие результаты. В процессе разработки программных средств почтовые сообщения и телефонные звонки на самом деле не компенсируют недостаток личного общения. Поэтому вы должны сочетать возможность удаленной работы с потребностью в работе совместной. Когда ответственные лица находятся в одном месте в одно время, на принятие решений у них уходят считан-

ные минуты. Если же они находятся в разных местах, на решение аналогичных проблем могут уходить часы, а то и целые дни.

Предоставляйте лучший инструментарий

В распоряжении любого программиста должен быть быстродействующий компьютер с максимально возможным объемом памяти. Кроме того, программисту нужна тестовая машина, воспроизводящая стандартные характеристики пользовательских компьютеров. Вполне вероятно, что в вашей компании наличествует сетевая инфраструктура, призванная обеспечивать поддержку разрабатываемых продуктов (веб-серверы, метафреймы Citrix Metaframe и т. д.). Соответственно, для тестирования результатов разработки эти продукты следует дублировать зеркалами. Иногда они могут использоваться совместно с группой тестирования, однако имейте в виду, что отделение сред разработки и тестирования от непосредственного производства себя оправдывает. Мне приходилось встречаться с компаниями, которые бездумно разрешали программистам напрямую редактировать веб-сайты путем удаленной загрузки файлов. Это самый неудачный метод, какой только можно себе представить. Он, конечно, удобен, но чрезвычайно опасен.

Если вашим сотрудникам приходится переходить с места на место (или сверхурочно работать дома), лучше всего приобрести для них ноутбуки. Сегодня они практически не уступают по своим возможностям настольным компьютерам, поэтому экономить не стоит. Не забывайте к тому же, что программисты предъявляют к своим машинам значительно более высокие требования, чем средние пользователи. Возможно, вам придется скорректировать принятую в компании политику технического обеспечения с учетом потребностей ваших подчиненных. Программистам нужно предоставить полномочия администратора в отношении прав доступа и конфигурирования их собственных машин. Если кто-то из них запутается в конфигурации, покажите, как решить проблему, которую он сам себе создал. Прибегать к услугам специалистов следует лишь в крайнем случае. Программисты, которые не знают, как сконфигурировать операционную систему или установить ее с чистого листа, просто недостаточно научены. Их уровень владения понятиями TCP/IP не должен уступать уровню системных инженеров.

В конце рабочего дня

Скорее всего, вы уже предвкушаете разговоры насчет наведения порядка на рабочем столе и перекладывания вещей с места на место. Это все хорошо, но мне хочется сказать кое-что более важное. Каких бы высот по части организованности вы ни достигли, чувства усталости и переутомления не избежать. Таков уж характер деятельности менеджера. Производство программных средств, руководство людьми и лидерские функции в их среде, равно как и все другие действия, осуществляемые руководителем, иногда приводят его в состояние прострации. Как бы то ни было, обобщая все вышесказанное, надеюсь, что вы запомните несколько принципов, способствующих достижению успеха.

- Допускают ли ваши собственные рабочие условия оперативность при превращении информации в действия? Если нет, постарайтесь систематизировать делопроизводство в соответствии с рекомендуемыми методиками в этой

области — так, чтобы, помимо просто руководства, вы могли взять на себя лидерские функции.

- Работайте над теми сферами своей деятельности, которые поддаются контролю, и смиритесь с тем, что некоторые из сфер неконтролируемы.
- Наладив активное взаимодействие с другими отделами, всеми силами способствуйте повышению организационного уровня компании в целом. Держите дела в своем отделе под контролем и будьте в этом отношении примером для других руководителей.

Большие задания следует дробить на более мелкие и лучше управляемые. Вооружившись этим подходом, можно с готовностью приниматься за решение глобальных задач. Иначе говоря, выводите вторичные задачи, которые предусматривают возможность решения в течение одного непрерывного промежутка времени. Так вы сможете организовать свои временные ресурсы и решать по одной задаче за раз. Ничто так не поднимает уровень самооценки, как наличие структурированного плана и его последовательное выполнение. Уверенность в своих силах, в свою очередь, приводит к достижению успеха, снижению стрессовых нагрузок и к удовлетворению собственной деятельностью, подпитываемой помощью в достижении результата окружающим вас людям.

Что дальше

В следующей главе мы поговорим о том, какую роль вы должны исполнять на совещаниях. Организационные навыки, проявляемые на совещаниях, вносят существенный вклад в достижении поставленных целей. В идеале совещания с другими группами (и, в особенности, с собственными сотрудниками) должны проводиться исходя из четко сформулированных целей и не менее определенных представлений о предполагаемых результатах. По мере ознакомления с материалом моей книги вы постепенно придете к мысли, что решить все проблемы сразу не получится. Вряд ли, к тому же, вам удастся извлечь из этой книги точные схемы действий в тех или иных ситуациях. Темы, которые я поднимаю, придется переосмыслить — так, чтобы выстроить материал согласно специфике исполняемых вами функций. Итак, вперед — читайте сердцем! Делайте с моим материалом все, что сочтете нужным, — я не обижусь.

Глава 5

Как вести совещания

Почему совещания нужно вести? Дело в том, что когда программисты, намереваясь обсудить ту или иную проблему, собираются вместе, их личностные качества имеют обыкновение вступать в конфликт, который, естественно, достижению поставленных целей совещания не способствует. Привыкнув заниматься кодированием, вы, скорее всего, перед проведением первого (или пятидесятого) совещания будете испытывать неприятные опасения[1]. Не беспокойтесь, эта глава, посвященная проведению совещаний, призвана повысить вашу подготовленность в этой области. Именно от вашей разумности зависит успех в деле ведения совещаний.

Технари вроде нас с вами часто страдают аллергией на совещания. Нам кажется, что, кроме потери времени, от них нет никакого толку, и значительно эффективнее лишний час повисеть над клавиатурой и попытаться написать приемлемый код. Как руководитель вы действительно ожидаете, что сотрудники будут проводить драгоценные рабочие часы за компьютером, но в то же время вы должны контролировать их деятельность. Совещания помогают координировать результаты ваших собственных усилий. Рассматривать совещания как неизбежное зло не следует, ибо они таковым не являются. Собираясь вместе со своими сотрудниками, можно обсуждать идеи друг друга и тем самым совершенствовать представления рабочей группы о выбранных стратегии и тактике.

Еженедельные совещания

Я рекомендую проводить совещания с сотрудниками каждую неделю в одно и то же время. Не стоит отлынивать от этих встреч, даже если вы плохо себе представляете,

---

[1] Получилось масло масляное — ведь опасения не бывают приятными. Не стоит, впрочем, излишне персонифицироваться — скорее всего, сотрудники вашей группы, общаясь с вами на совещании, испытывают аналогичные чувства. Вряд ли вас это успокоит — помните, что за лидерство тоже нужно платить.

что на них можно обсудить. На самом деле предмет для обсуждения найдется — нужно только каждую неделю придерживаться на совещании четкой повестки дня.

- Что вы делали на прошлой неделе?
- Что вы собираетесь делать на этой неделе?
- Что мешает вам выполнить ваши задания в назначенное время?

Заставьте своих сотрудников приносить на совещания списки поставленных перед ними задач и просите их каждый раз отвечать на эти три простых вопроса. При формулировании заданий вы устанавливаете для них сроки завершения, не так ли? Задания программистам нужно давать в письменном виде, даже если текст задания ограничивается высокоуровневым описанием очередной характеристики программного продукта. Зачастую выполнение основного задания предполагает предварительное решение ряда вспомогательных задач. На все эти задачи — основные и второстепенные — нужно обращать внимание; вы должны постоянно напоминать сотрудникам о том, что от них требуется. В зависимости от уровня руководства проектом, который вам доступен и которому вы симпатизируете, список задач может выглядеть совершенно по-разному — в диапазоне от простого описания задачи с указанием срока ее решения до комплексной временной схемы проекта с детальной росписью подзадач для каждого значимого объекта поставки. О назначении заданий я говорил в предыдущей главе. Попробуйте придумать собственный метод составления списков заданий. Адаптируйте его к индивидуальным характеристикам компании и к собственному стилю руководства. Наличие списков заданий должно стать обязательным условием проведения еженедельного собрания персонала.

Во время совещания делайте акцент на понятии «объектов поставки». Этим понятием обозначаются характеристики или продукты, поставляемые вашими подчиненными программистами для публичного потребления. Избегайте призрачных намеков на результат. Не надо говорить программистам о том, что они должны «исправить ошибки» или «внести некоторые доработки».

Во время совещания делайте акцент на понятии «объектов поставки». Этим понятием обозначаются характеристики или продукты, поставляемые вашими подчиненными программистами для публичного потребления. Избегайте призрачных намеков на результат. Не надо говорить программистам о том, что они должны «исправить ошибки» или «внести некоторые доработки».

Если в вашей компании есть мощная группа с сильными маркетинговыми ресурсами, в задачу которой входит исчерпывающее описание продукта, ваши функции заметно упрощаются. Повторюсь: на совещаниях, вместо того чтобы спорить с программистами о хитросплетениях кода, вы должны проверять сотрудников и обеспечивать их готовность в срок решить задачи, поставленные в связи с производством программного продукта. С другой стороны, иногда легкомысленное отношение к коду способствует его более успешной разработке; поэтому старайтесь не предстать в глазах сотрудников тираном.

Достоинство предложенной мной программы совещания состоит в том, что она проста, предполагает диалог с сотрудниками и обеспечивает возможность выявления трудностей, с которыми они сталкиваются. Заставляя программистов фор-

мулировать достигнутые цели и ставить перед собой цели на неделю, вы укрепляете в их сознании мысль о том, что они работают ради создания конкретного продукта. С другой стороны, вам нужно, чтобы программисты, копаясь в деталях реализации, держали в голове все вспомогательные задачи, направленные на решение более общей (высокоуровневой) задачи. Сначала полезно определить в контексте проекта высокоуровневую задачу — например, «реализовать модуль, выражающий функцию *X*», а затем, исходя из имеющихся ресурсов, сформулировать вспомогательные, зависимые задачи. В процессе администрирования за этим также придется следить. Если бы мы, руководители, могли держать в голове все детали, связанные с реализацией любой конкретной характеристики программного продукта, было бы, конечно, замечательно; впрочем, в таком случае мы утратили бы потенциал делегирования и фактически отказались бы от обращения к интеллектуальным ресурсам и техническим навыкам своих сотрудников. Скажем подругому: пытайтесь держать в голове как можно больше деталей, и если ваша команда состоит из профессионалов с достаточной мотивацией, они помогут вам справиться с задачей.

В конце концов, над результатом работает вся группа, а ваша цель заключается в том, чтобы ставить планки и проверять результаты.

Время от времени к своей повестке дня я прибавлял разбор литературы. Выберите добротную книгу о методиках программирования, принципах проектирования или по какой-либо другой теме, связанной с будущим развитием нашей отрасли, и попытайтесь совместно с подчиненными разобраться в ее материале. Попросите каждого выступить на следующем совещании с устным докладом по ее содержанию. Создайте атмосферу университетского семинара, только не пытайтесь казаться слишком серьезным. Эта методика здорово стимулирует потребность в дальнейшем обучении.

Не сомневаюсь, что при необходимости вы учтете этот мой опыт; однако остерегайтесь растягивать совещания и обсуждать предметы слишком детально. Дело в том, что при продолжительности более 45 минут результат совещания может оказаться отрицательным. Совещание сотрудников способствует углублению интеграции в команде и предоставляет возможность специалистам, столкнувшимся с особо трудными задачами, обращаться к коллективному интеллекту. К тем, кто не справляется со своей работой в срок, следует относиться довольно жестко. Заставьте их объяснить остальным членам группы, по какой причине они опаздывают с решением поставленной задачи. Впрочем, не переусердствуйте в своих манипуляциях с групповым поведением. Публичное унижение сложно признать эффективным средством повышения производительности — поэтому, критикуя сотрудников, будьте сдержанны; в том же, что касается похвал (по крайней мере, на публике), не скупитесь.

Особое значение совещания персонала приобретают при приближении срока завершения работы над кодом. В такие моменты внимание группы следует обратить на особо нудные задачи и внесение в текст программы последних корректив — эти вопросы всегда необходимо решать с особой тщательностью. Проводя еженедельные совещания сотрудников, вы сможете поддержать их внимание, когда на подходе будут наиболее критичные этапы процесса разработки.

Каждую неделю при проведении совещаний будьте готовы к тому, что сотрудники станут оценивать ваши лидерские качества. Не стоит забывать, что, проводя

еженедельные совещания, вы нарабатываете соответствующий навык. Полжизни мы работаем на публику — а если серьезно, то, что бы там ни говорил Эмерсон (Emerson)[1], мне сложно представить себе ситуацию, в которой последовательный человек выглядел бы глупо. Еженедельно выискивая пути углубления кооперации между сотрудниками, вы формируете полноценную команду. Не стоит препятствовать проявлению соревновательного принципа — впрочем, в этом контексте вы должны играть уравновешивающую роль. Время от времени сотрудников следует ненавязчиво сдерживать или, напротив, побуждать к активным действиям. Если вы хорошо разбираетесь в своих подчиненных, то те действия корректирующего характера, которые вам предстоит предпринять, подскажет ваша интуиция. Эта роль напоминает позицию родителей по отношению к своим подросткам-отпрыскам (слава Богу, на эту тему мне писать не придется).

Проектные совещания

Будь вы хоть во сто крат талантливее лучшего программиста своей группы, скорее всего, всех ваших навыков, не говоря уже о времени, не хватит, чтобы самостоятельно реализовать все характеристики разрабатываемых программных продуктов. Даже если бы у вас нашлось время, разрабатывать все самому не имеет особого смысла — ведь тогда коллеги-программисты не смогут ощущать себя соучастниками творческого процесса. Иногда на проектных совещаниях нам приходится иметь дело с социально пассивной группой[2], которую неплохо бы превратить в команду активных, мотивированных и рвущихся к действию специалистов, готовых проектировать и реализовывать очередные характеристики вовремя и практически безошибочно. Такого рода индивидуальные особенности сотрудников зачастую довольно причудливо проявляются в контексте поведения группы в целом. Как настроить людей, с которыми приходится работать, на позитивный лад? Кто знает — может быть, вам и удастся достичь цели колдовскими чарами и материальным поощрением, но на самом деле эти способы неэффективны. Что вам действительно нужно, так это разум опытного психолога и душа дипломата-карьериста. Добро пожаловать в чрезвычайно запутанный, но в не меньшей степени притягательный мир группового поведения!

Проектировать программные продукты самому неразумно — ведь тогда коллеги-программисты не смогут ощутить себя соучастниками творческого процесса.

Так как же стать блистательным лидером проектной группы, если в данный момент вы таковым не являетесь? Без мыслительной деятельности и постоянной

---

[1] «Дурацкий принцип последовательности есть прерогатива ограниченных умов, побуждаемых не менее ограниченными политиками, философами и священниками. В оковах последовательности широкой душе негде развернуться...», — Ральф Уолдо Эмерсон (Self-Reliance, 1841).

[2] Я отнюдь не утверждаю, что все программисты социально пассивны, — мы просто, скажем так, особенные, и именно в этом кроется наша индивидуальность и благодаря этому мы способны проводить долгие часы в умственном напряжении, размышляя о том, как решить поставленные программные задачи.

практики ничего не выйдет. Рассмотрим, однако же, некоторые проблемы и явления, наблюдаемые на стандартном проектном совещании.

- Ваша задача — сделать так, чтобы все функции были корректно спроектированы и впоследствии качественно разработаны.
- У каждого участника группы могут быть собственные представления относительно проектного решения одной и той же функции.
- Программисты склонны проектировать лишь те функции, которые они могут или хотят разрабатывать.
- У некоторых программистов могут быть планы, отличные от ваших; трудно выявляемые, такие бунтари способны привести к саботажу совещаний.
- Единодушная поддержка сотрудниками высказанных вами идей (если только они не объективно гениальны) наводит на грустные размышления.
- В процессе проектирования вы обязаны пытаться достичь консенсуса — это трудная задача, но усилия, поверьте мне, окупаются. Достичь компромисса несколько легче, но если кто-то выступит со своей вымученной идеей и не получит поддержки, вы рискуете создать себе врага.

Совещания либо осложняют существующую в группе разработчиков ситуацию, либо разрешают кризис. В каком направлении пойдете вы, всецело зависит от вашей воли. Силами наличествующего персонала и с учетом предъявленных требований вы должны создать работоспособные спецификации, на основе которых впоследствии можно будет писать качественный код.

В большинстве компаний достичь этой цели довольно сложно. Слава Богу, моя книга — не более чем скромное руководство, и я совсем не претендую на то, чтобы осветить все мыслимые вопросы. (Молодец, Хэнк! Избавился от ответственности.)

Шучу. Если серьезно, обратите внимание на следующие рекомендации.

- Вы должны знать сильные и слабые качества участников своей команды. Не менее важно осознавать собственные достоинства и недостатки. Именно об этом я говорил в первых трех главах книги.
- Документацию с требованиями имеет смысл разбить на разделы по схожей функциональности, допускающие решение средствами объектно-ориентированного проектирования.
- Делайте заметки. Если можете себе это позволить, заведите электронное табло. Зафиксируйте совещание на видеопленке, оцифруйте ее, а после создания первого черновика проектной спецификации покажите отснятый материал членам группы.
- Подключите ноутбук к большому монитору и демонстрируйте сотрудникам группы примеры готового кода, указывая на те его аспекты, которые считаете удачными или, напротив, неудачными.
- Уберите из помещения, в котором проводите совещания, телефонный аппарат. Этого можно не делать лишь в том случае, если во время совещания вы собираетесь обратиться к какому-нибудь удаленному абоненту за свежими идеями.

- Работайте в удобном и изолированном от шума помещении. Делайте перерывы, чтобы усталость организмов присутствующих не мешала деятельной работе их мозгов.
- Составьте план действий на каждый день; при необходимости вносите в него коррективы и отталкивайтесь от него в деле реализации проектных заданий. Отведите последний день на критический обзор и обобщение результатов совещания.

Как я только что сказал, вести заметки совершенно необходимо. Если вы регулярно выходите к электронному табло с предложением новых идей по проекту, что-то писать на бумажке становится довольно проблематично. В таком случае имеет смысл попросить одного из сотрудников выполнять эту функцию за вас. Представляете, как будет обидно, если, проведя плодотворное совещание на прошлой неделе, вы забудете о принятых на нем решениях уже через несколько дней! Для человека, которому вы намерены поручить протоколирование совещания, можно составить специальный шаблон. Пример такового показан на рис. 5.1.

Несколько замечаний по поводу шаблона. Большая часть информации, которая в нем фиксируется, очевидна. Самая пространная область шаблона — «Замечания» — предусматривает запись информации в свободной форме. В ней можно, скажем, начертить блок-схему (если электронное табло неисправно) или просто зафиксировать какие-то идеи, которые во время совещаний часто появляются спонтанно. Раздел «Влияние на проектное решение» тоже довольно важен. Почти всегда артефакты проектирования оказывают влияние на существующее программное обеспечение, равно как и на другие аспекты деятельности компании. Вот эти-то варианты воздействия вам и предстоит зафиксировать. Область «Необходимые действия» ориентирует в последующих действиях, направленных на реализацию проектного решения. Ведь действительно совещание без реализации принятых решений на практике есть не более чем потеря времени. Кроме того, имеет смысл составить краткое письменное резюме проблем, обсуждавшихся на совещании, и присовокупить его к базе знаний группы разработчиков. Эта информация может понадобиться тем сотрудникам, которые по той или иной причине не смогли присутствовать на совещании, а также специалистам, только что присоединившимся к вашей группе и считающим необходимым ознакомиться с историей проекта.

Совещание без реализации принятых решений на практике есть не более чем потеря времени.

В целях соблюдения четкости формулировок постарайтесь, чтобы размер шаблона не превысил одной страницы. Если относительно какого-то конкретного проектного решения увеличить размер шаблона все-таки потребуется, возьмите дополнительный лист и соответствующим образом пронумеруйте его. Полагаю, вы поняли, что я имею в виду. Не сомневаюсь также, что вы сможете придумать более удобный метод. Главное — сделать так, чтобы при переходе на следующий этап проектирования (а именно к спецификации проектного решения) вы не опирались лишь на свою память. Ну ладно, я закругляюсь с этой темой — все-таки мою

книгу сложно причислить к комплексным исследованиям по управлению проектами и, тем более, к монографиям по объектно-ориентированному проектированию.

Продукт/Версия _____ /_____ **Дата** _____

Характеристика/Функция _____

Замечания

Влияние на проектное решение

Родственное программное обеспечение	Инфраструктура	Другие отделы

Необходимые действия

Что	Кто	Когда

Рис. 5.1. Вариант шаблона для записей на совещаниях

Не углубляясь в дебри программного проектирования, отмечу один момент, касающийся персонала. Вы должны осознать, что несмотря на ваши функции руководителя, совершенно не факт, что из всех сотрудников группы у вас самые яркие архитектурные способности. Как вы знаете, программисты все разные, и зачастую в одном человеке мирно уживаются характеристики разных типов. (Вспомните мою классификацию пород, приведенную в главе 1.) Если вы отдаете себе отчет в том, что ваши возможности по части глобального архитектурного мышления весьма ограниченны, выделите среди своих сотрудников тех людей, которые способны достигнуть в этом отношении более серьезных результатов, и позвольте им принимать в процессе как можно более деятельное участие. Следует быть готовым к непреодолимому желанию тоже поучаствовать в процессе генерации идей, однако имейте в виду, что ваша основная задача — справляться с отбором предлагаемых идей, а не генерировать свои. Быть пропагандистом хороших идей, уверяю вас, обычно не менее почетно, чем быть их генератором. В этой области вам придется пустить в ход свои дипломатические способности.

Вы говорите, что у вас нет дипломатических навыков? Что я могу сказать? Учитесь! Дипломатия — это искусство выслушивать, прежде чем говорить, думать, прежде чем предлагать, и постоянно искать консенсус.

Дипломатия — это искусство выслушивать, прежде чем говорить, думать, прежде чем предлагать, и постоянно искать консенсус.

Задача не из простых, кто бы спорил, но ведь деньги, которые вам платят, надо отрабатывать! Рекомендую прочесть несколько книжек по поводу формирования команды (см. библиографию). Но это не главное. Основная предпосылка к достижению желаемого результата — сильное желание. Стремление добиться общего для всех участников группы успеха вы должны ставить выше искушения выиграть случайный спор. Запомните, выиграть спор невозможно! В условиях командной работы все вместе либо выигрывают, либо проигрывают.

Я упомянул, что основной целью проектного совещания является выработка консенсуса. Я совершенно не имею в виду голосование по предложенным идеям. Демократия — система, прекрасно подходящая для наций, — не приживается в техническом проектировании[1]. Отдав предпочтение идеям одного человека и, соответственно, отказавшись от предложений другого, вы не добьетесь продуктивного мышления. Консенсус достигается за счет синтеза идей. Торг по принципу «я соглашусь на твою характеристику, если ты согласишься на мою», здесь неуместен. Что я понимаю под синтезом? Это процесс, в ходе которого путем проработки конкурирующих идей выкристаллизовываются их наилучшие качества. С вашей стороны для достижения этой цели требуется терпение и настойчивость. Ведь вы как руководитель должны стремиться к тому, чтобы вычленить наилучшее из всех возможных проектных решений. Это та цена, которую приходится платить за успех. В своем потрясающем сборнике статей по кадровому обеспечению Ларри Константайн высказывает следующие соображения:

> «Синтез приводит к появлению оригинальной концепции, в которую входят сущностные характеристики всех представленных идей и предложений... Консенсус, построенный на основе синтеза, не только включает в себя лучшие из предложенных альтернатив, — он, как правило, вводит новые характеристики и возможности, являющиеся логическим продолжением объединения высказанных идей»[2].

КОШАЧЬИ РАЗБОРКИ — ЧУЖАК

Работать с Полом было чрезвычайно трудно. Он меня ужасно раздражал. То обстоятельство, что директор переманил его от нашего главного конкурента, назначив ему зарплату больше моей, а затем поручил мне им руководить, ничуть не способствовало смягчению наших отношений. Ну и что, что он доктор физических наук и единолично написал

[1] Мне очень понравилась фраза, которую в фильме «Crimson Tide» Джин Хэкман сказал Дэнзелу Вашингтону: «Мы должны не заниматься демократией, а оберегать ее». Аналогичным образом, проводя проектное совещание, лучше воздерживаться от реверансов в пользу той или иной обсуждаемой идеи — как бы красноречиво ее ни защищали приверженцы. Ваша цель заключается в том, чтобы поощрять активный и плодотворный мыслительный процесс.

[2] Larry L. Constantine, The Peopleware Papers (Upper Saddle River, NJ: Yourdon Press, 2001), p. 10.

потрясающую программу, которая заставила нашу компанию заметно потесниться на рынке? С ним было невозможно работать. Он был полностью уверен в своей неизменной правоте и воспринимал коллег как молокососов. Естественно, собственные представления о совершенстве были для него единственной точкой зрения на все предполагаемые функции программы. Он не хотел ничего знать ни о компонентной разработке, ни о программировании в команде. Передо мной стояла невыполнимая задача — задействовать Пола в процессе разработки нового поколения программных продуктов, с помощью его идей побудить к действию остальных специалистов и при этом умудриться сделать так, чтобы они тоже почувствовали свою причастность к проектному решению.

Итак, я принялся за планирование первого проектного совещания, на котором должен был появиться Пол. По моему плану на совещании предполагалось описать те архитектурные методики, которые мы применяли ранее, и показать их в роли проводника нашей компании в будущее. У Пола была другая точка зрения. Он заявил мне буквально следующее: «Ральф, неужто ты действительно думаешь, что на этих изношенных временем идеях можно построить революционный продукт?» Вопрос был, конечно, риторический — по крайней мере, мне хватило мозгов, чтобы прийти к тому же выводу. Что же мы имели после недели проектных совещаний? А ничего, кроме критических комментариев Пола и полного недоумения остальных сотрудников, искавших поддержки с моей стороны. Впрочем, это обстоятельство не повергло меня в уныние. И хотя неделька выдалась совершенно ужасной, я не придумал ничего лучшего, чем позволить Полу спокойно заниматься разработкой, надеясь, что все, в конечном итоге, образуется.

Через месяц я пришел к директору и заявил, что, не имея возможности распоряжаться рабочим режимом Пола (его зарплатой, вопросами его повышения, увольнения и проч.), я не могу им управлять и, соответственно, не беру на себя ответственность за результаты его дальнейшей деятельности. К сожалению для меня, директор согласился с этими доводами; только после этого я осознал, в какой кошмарной ситуации оказался. Получалось, что если результаты деятельности Пола окажутся неудовлетворительными, винить в этом можно будет только меня. Была и другая трудность — я начал нравиться Полу. Он-то думал, что я изолировал его от группы по той причине, что оценил его способности, а совсем не потому, что с учетом его кошмарного характера у меня не было другого выхода. Как впоследствии выяснилось, никаких особых способностей у Пола не было. Ему просто один раз повезло. Как я узнал после многочисленных и продолжительных ужинов с этим деятелем, за долгие годы он сменил уйму должностей и успел позаниматься самыми разными делами, для всех он оказывался чужаком.

Конец истории взаимоотношений Ральфа и Пола неутешителен. В конце концов, Ральфу пришлось уволить Пола — по той лишь причине, что выхода его следующего чудо-продукта из стадии альфа-тестирования никто так и не дождался. Остальные члены группы, ощущая свою полную невостребованность, замкнулись в себе, и я их прекрасно понимаю. Большую часть своего времени Ральф тратил на то, чтобы изолировать Пола от каких бы то ни было других контактов с остальными специалистами, поскольку совещания тот неизменно превращал в сеансы разгромной критики их работы. Из этой истории мы можем сделать несколько выводов.

- Прошлые достижения отдельного сотрудника группы совершенно не гарантирует успешной деятельности группы в целом.
- Формированием группы должен заниматься только ее руководитель. В этом отношении он должен пользоваться полным доверием своего начальника.
- Нельзя оценивать потенциал сотрудника, отталкиваясь только от его последних достижений. Необходимо тщательно изучить весь опыт его работы.
- Некоторые люди просто не способны работать в команде, и не стоит заставлять их учиться этому. Возможно, они просто не подходят для работы в вашей компании.

Беседы один на один

Время от времени имеет смысл общаться с каждым из сотрудников в отдельности. Разумно также отвести некоторое время на консультирование новичков и попытки помочь тем квалифицированным специалистам, которые, несмотря на квалификацию, завязли в своем проекте. К подобным беседам следует относиться с не меньшей серьезностью, чем к любым другим разновидностям совещаний. Планируйте их, делайте заметки и учитывайте специфику межличностных отношений с конкретным сотрудником. Результатом беседы должна стать выработка плана действий. Положительных эмоций, почерпнутых от общения с хорошим человеком, в данном случае недостаточно.

К встречам один на один следует относиться с не меньшей серьезностью, чем к любым другим разновидностям совещаний. Планируйте их, делайте заметки и учитывайте специфику межличностных отношений с конкретным сотрудником. Результатом беседы должна стать выработка плана действий. Положительных эмоций, почерпнутых от общения с хорошим человеком, в данном случае недостаточно.

Попытайтесь собраться (как вы понимаете, дисциплина — это одно из тех качеств, которое вам как лидеру постоянно придется совершенствовать) и выделите один день в неделю на плотное взаимодействие с каждым из подчиненных. Не всем требуется консультация один на один, но, с другой стороны, если вы будете помогать своим сотрудникам, они станут более серьезно относиться к поставленным вами задачам и, осознавая свою значимость для вас, постараются их решить. Нужно сделать так, чтобы ваши цели стали их целями. Уделяя сотрудникам время, вы сможете выработать в них чувство групповой ответственности за решение поставленных перед вашим отделом задач. Эта цель лучше всего реализуется при общении один на один.

Специалисты — это ваш самый ценный ресурс. С другой стороны, они способны создавать самые неприятные проблемы. Следовательно, построение устойчивых, профессиональных отношений с подчиненными есть залог вашего преуспеяния в роли лидера. Остерегайтесь переводить отношения с сотрудниками на личностную основу. Это довольно трудно — в конце концов, любое общение между двумя людьми проходит на личностном уровне. Тем не менее в общении с сотрудниками вы должны поддерживать профессиональную направленность, подкрепляя тем самым свое лидерское положение. Спору нет — друзья нужны любому, однако в бизнесе такие отношения не всегда хороши. В дело вмешивается слишком много внешних обстоятельств — хотя бы тот факт, что благополучие подчиненных всецело зависит от вас. Если в какой-то момент вам придется принять решение об увольнении сотрудника, с которым вы установили личные дружеские отношения, такой поступок рискует оказаться слишком трудным. Если же отношения с ним строго профессиональны, вы сможете проявлять большую объективность.

Такие вопросы, как повышение заработной платы, повышение по службе, решаются исключительно в ходе бесед один на один. Если в вашей компании утверждена формальная процедура критического анализа, у вас есть шанс остаться объективным. Следует отдавать себе отчет в том, что любой критический анализ

субъективен по своей сути, что, впрочем, нисколько не умаляет важности этого процесса. Исполняя роль лидера, вы должны постоянно наблюдать за результатами сотрудников. Попытки вспомнить о прошлых достижениях и неудачах сотрудника при проведении критического анализа его текущей деятельности — не слишком удачный подход. Один из вариантов оценки продуктивности сотрудника (либо на бумаге, либо в электронном виде) предполагает еженедельный сбор информации о результатах его работы. Если вы ведете личные дела, отмечайте в них успехи и затруднения каждого работника. В частности, в этих документах следует указывать адрес электронной почты сотрудника, динамику соблюдения или, напротив, нарушения им сроков сдачи работ, участие в выработке проектных решений, предложения по решению проблем.

Помимо прочего, во время встреч с сотрудниками «один на один» вы получаете возможность отслеживать работу над проектом и препятствия, с которыми подчиненные сталкиваются в своей деятельности. Этот процесс можно формализовать. К примеру, можете завести стандартный опросник и попросить сотрудников группы раз в неделю заполнять его. Информация, содержащаяся в анкетах, позволит вам выявлять трудности и своевременно направлять усилия на их преодоление. При просмотре анкеты конкретного сотрудника пригласите его к себе в кабинет и обсудите варианты разрешения возникших проблем. Решать проблемы по мере их появления, на ранних этапах значительно эффективнее, чем биться с проектом, разъедаемым проблемами, не выявленными вовремя.

Совещания с другими группами

Итак, вы получили новую должность. Теперь вам предстоит существенно расширить границы своего общения. На эту тему есть иллюстрация Скотта Адамса (Scott Adams) — обратите внимание на позицию руководителя (рис. 5.2).

Рис. 5.2. Программист, только что узнавший о повышении

Надеюсь, что вы запуганы чуть меньше, чем изображенный на рисунке программист. Мы, технари, сталкиваясь с необходимостью взаимодействия с группами разработчиков, с которыми в обычных условиях не общаемся, чувствуем себя не слишком уверенно. Будьте готовы к тому, что «обычный» круг общения начнет расширяться. Далеко не все совещания, которые вам предстоит провести, предполагают

присутствие исключительно подчиненных программистов. На них, скорее всего, будут приходить сотрудники нетехнического профиля, равно как и специалисты в тех технических дисциплинах, в которых вы не слишком смыслите. Определенного внимания к себе будут требовать сотрудники, ответственные за формулировку бизнес-требований, люди из отделов поддержки и тестирования, специалисты по проверке качества, финансовые сотрудники и многие другие.

Как вести себя на таких совещаниях? На некоторые из них придется приходить в официальном наряде — поэтому пополните свой гардероб хотя бы одним хорошим костюмом. Помните, люди не вашего круга ожидают, что вы покажете себя немного странноватым; постарайтесь, впрочем, держать эту наигранную придурковатость в рамках стереотипа творческой личности и сделайте так, чтобы люди, привыкшие считать доходы и далекие от логических выражений, воспринимали вас адекватно.

Всегда обрисовывайте ваш отдел в идиллических тонах. Никогда не сваливайте вину за срыв сроков на своих подчиненных. Такую позицию очень не любят, и подобными действиями вы только подогреваете нашу и без того пожароопасную индустрию, в которой программисты при оценке своих попыток соблюдения контрольных сроков проявляют себя убежденными идеалистами. Мы можем сколь угодно пребывать в уверенности, что программирование есть искусство (а это действительно так). При этом не стоит забывать, что ваш начальник мыслит по-другому; по его мнению, программирование есть наука с присущей этому понятию предсказуемостью. Выслушивая похвалы, считайте, что вам повезло, но реагируйте на них скромно и достойно; делайте акцент на том, что если бы не ваша команда, никаких достижений не было бы в помине. Хотите высказать сотруднику свои претензии — ради бога, но только в приватной обстановке.

Никогда не сваливайте вину за срыв сроков на своих подчиненных. Такую позицию очень не любят, и подобными действиями вы только подогреваете нашу и без того пожароопасную индустрию, в которой программисты при оценке своих попыток соблюдения контрольных сроков проявляют себя убежденными идеалистами.

Успешное взаимодействие с остальными подразделениями компании заставит сотрудников уважать вас. Уважению не будет предела, если вы позволите им не присутствовать на длительных и иногда утомительных совещаниях, которых вам самому никак не избежать.

Одна из основных ваших задач в процессе взаимодействия с другими группами заключается в том, чтобы обеспечить своевременное получение бизнес-требований в как можно более завершенном состоянии. Разбухание области действия, как известно, начинается со скверно определенных требований. Ситуация ухудшается, когда программисты, столкнувшись с плохим описанием продукта, начинают фантазировать, и с этого момента разрастание области действия уже практически невозможно контролировать. Этапу описания продукта следует уделять не меньше времени, чем этапам проектирования и программирования. Такие временные затраты окупаются — дело в том, что, чем более комплексной информацией вы обладаете по части коммерческих приоритетов компании, тем лучше себе представляете, какой продукт ей необходим для того, чтобы занять достойную позицию на рынке.

Еще несколько слов о требованиях. Будучи программистом, вы, естественно, любите разрабатывать программные продукты. Отталкиваясь от идеи, вы преобразуете ее в виртуальную реальность. Согласитесь, есть некое волшебство в создании выверенного пользовательского интерфейса, который, к тому же, грамотно состыкован с прикладной частью. Скорее всего, лучшим из всех созданных вами программных продуктов был тот, относительного которого вы четко представляли себе бизнес-требования. Думаю также, что как программист вы получили от разработки этого приложения больше всего приятных минут. Постарайтесь донести это чувство до группы описания продукта, и по мере сбора требований очерчивайте ее сотрудникам пределы возможного. За счет своих технических способностей вы сможете на корню избавляться от всех неудачных идей; навыки же сотрудников группы описания продукта в области бизнеса помогут вам генерировать новые идеи. Учитесь организовывать сотрудничество технологии и бизнеса; при этом не забывайте, что доминирующую роль в этом союзе играет именно бизнес. Вряд ли вам как технарю это понравится, но такова реальность, и подход, о котором я говорю, себя окупает.

Ретроспективные совещания

Будем надеяться, что поминки проектов вам проводить не придется. Совещания, тип которых обозначен в заголовке, не должны представлять собой арену для выражения недовольства; на самом деле это лишь формальный способ обучения на собственном опыте. Как писал Норман Кит (Norman Keith):

> «Степень эффективности и успеха ретроспективных совещаний обусловливается безопасностью сотрудников. Под безопасностью я имею в виду защищенность сотрудников от критики в рамках их группы. Лишь при таком условии они смогут обсуждать собственные результаты и даже признаваться в том, что поставленных целей можно было достичь по-другому, более оптимальными способами — иначе говоря, учиться анализировать выполненные проекты. Безопасность необходимо культивировать и поддерживать. Несмотря на то, что, по большому счету, безопасность выражает ответственность всех участников ретроспективного совещания, его руководитель призван создавать условия для безопасности, следить за ее поддержанием и контролировать это ощущение. Чтобы сотрудник чувствовал себя в безопасности, он должен быть уверен, что за проявленную честность он не получит по ушам (например, не нарвется на отрицательную оценку во время следующего критического обзора). В ходе ретроспективного совещания не обойтись без постоянного и должным образом поощряемого взаимодоверия»[1].

Проведя ретроспективное совещание в «безопасном» формате, вы получаете хорошие шансы почерпнуть довольно существенные сведения относительно недавно завершенного проекта. Эти знания, в свою очередь, позволят вам усовершенствовать процесс разработки. Необходимость соблюдения на ретроспективных совещаниях ощущения безопасности связана с тем, что иногда сотрудникам на них приходится отвечать на неприятные вопросы.

---

[1] Norman L. Kerth, Project Retrospectives (New York: Dorset House Publishing, 2001), p. 7.

В ходе ретроспективного совещания, помимо прочего, полезно определиться с ответами на нижеследующие вопросы.

- Насколько четко была сформулирована спецификация продукта? Другой вариант того же вопроса: не случилось ли так, что в силу многократного пересмотра спецификации этап проектирования пришлось отложить на слишком долгий срок?
- Нашлось ли у вас время на макетирование проектного решения или же вы сразу приступили к кодированию?
- Трудно ли было расширять существующую архитектуру новыми функциями?
- Внес ли руководитель проекта весомый вклад в его успешную реализацию? Как можно оценить его организованность, компетентность и готовность к участию в проекте?
- Если бы вам представилась возможность написать тот же код снова, сделали бы вы что-нибудь по-другому?
- Находились ли в вашем распоряжении все программные инструменты, необходимые для решения поставленных задач?
- Как вы думаете, какие составляющие процесса разработки имеет смысл изменить?

Последний вопрос, естественно, открывает возможности для обсуждения широкого круга разнообразных проблем. Такие дискуссии только приветствуются — правда, проводить их следует как можно ближе к предмету обсуждения, то есть к выполненному проекту, и поощрять конструктивные предложения.

Телеконференции

Телеконференции отличаются от традиционных совещаний лишь тем, что не предполагают применения языка жестов и обычно проводятся в соответствии с четким планом в приличных акустических условиях. Впрочем, стоит заметить, что невозможность использования языка жестов существенно снижает эффективность совещаний. Несмотря на это, географические факторы иногда принуждают руководителей проводить совещания именно по телефону. В нашей отрасли они на сегодняшний момент довольно распространены. Если во время совещания планируется обсудить те или иные зрительные образы, не забудьте предварительно разослать экземпляры соответствующих изображений и попросите участников с ними ознакомиться. Не нужно зачитывать на совещаниях какие бы то ни было документы — ваши сотрудники взрослые люди, они умеют читать; соответственно, проводя время таким образом, вы его попросту убиваете.

При проведении телеконференций опасайтесь безоговорочно полагаться на принцип «молчание — знак согласия». Иначе говоря, если на принятое вами решение или донесенную вами информацию не поступает никаких комментариев, считается, что слушатели согласны с вашей позицией. В целом, действие этого принципа следует оценивать положительно — в конце концов, он способствует проявлению участниками совещания активной позиции. Поскольку в данных условиях вам не известно ни выражение лица, ни мимические реакции собеседни-

ков, молчание действительно правомерно воспринимать как знак согласия. Тем не менее применимость этого принципа во многом зависит от специфики той группы людей, с которыми вы общаетесь. Если вам приходится проводить телеконференции с людьми, с которыми вы ни разу не встречались, принимать молчание за согласие как минимум опрометчиво. Совещание с незнакомыми людьми, построенное на селекторной связи, эффективным сделать довольно трудно. По этой причине, если вы можете себе позволить проводить видеоконференции, лучше всего остановиться именно на них.

Растягивать телеконференцию более чем на час не имеет смысла. Чрезмерная продолжительность такого совещания свидетельствует либо о его перенасыщенности, либо о неудачной повестке дня. Начинать совещание следует в оговоренное время, причем ждать, пока к нему подключатся все предполагаемые участники, не нужно. Если уж они пропускают важные сведения из-за собственной медлительности, пусть учатся на своих ошибках. Не стоит также и опаздывать с открытием совещания — во-первых, тем самым вы проявляете неуважение к собеседникам, во-вторых, демонстрируете непрофессионализм, в-третьих, подаете косвенный сигнал о том, что обсуждаемые вопросы на самом деле не так уж важны.

Помните, что при проведении телеконференций вашим единственным инструментом оказывается голос. Пользоваться им стоит разумно и осмысленно. Фиксируйте на бумаге основные моменты совещания (или поручите эту функцию одному из своих сотрудников); не забывайте также и о том, что после окончания совещания вы должны как можно быстрее отправить всем его участникам стенограмму с четкими указаниями к действию. Во время телеконференций следует избегать лишней болтовни. В умеренном остроумии нет ничего предосудительного, однако излишне усердствуя в этом отношении, вы рискуете растянуть совещание.

Время между совещаниями

Не слишком удивляйтесь тому, что я сейчас предложу. Сотворите про себя «легенду». Она обязательно должна основываться на реальных фактах, однако если окружающие вас люди будут проявлять стремление к приукрашиванию вашей преданности работе, не мешайте им в этом. Впрочем, каких-то особых шагов в этом направлении предпринимать не стоит. Пусть за вас говорят результаты. О чем должна быть ваша легенда? Всего-навсего о том, что вы последовательно придерживаетесь принципов эффективного лидерства — не больше и не меньше.

Не путайте преданность с плохим планированием. Большинство менеджеров работают больше стандартных 40 часов в неделю. При этом переработка не всегда свидетельствует о том, что руководитель предан своей деятельности. Вполне возможно, он просто не слишком организованный человек, не справляющийся со своими обязанностями в разумное время. Кроме того, если вы пренебрегаете личной жизнью и ставите работу во главу угла, вполне вероятно, что в легенде вы предстанете отъявленным фанатиком. Работая больше, чем требуется, вы вольно или невольно заставляете окружающих вас людей также стремиться к увеличению своего рабочего времени. Иногда это действительно необходимо. Однако, делая из этого привычку, имейте в виду, что ваши лидерские качества развиты недостаточно

и ваш персонал страдает. Как я говорил в предыдущей главе, контролировать рабочее время и определить роль работы в контексте своего существования вполне в ваших силах.

Наверное, я излишне перехожу на личности, но факт остается фактом — руководитель, действующий эффективно, оказывает на своих подчиненных определенное влияние. То, каким будет это влияние, зависит от вашей меры ответственности. О том, каким вы останетесь в памяти людей, с которыми работаете, нужно думать не меньше, чем о взаимоотношениях внутри семьи. Если у вас целостный характер, ваш публичный образ станет отражением личностных характеристик. Поймите меня правильно: я не призываю к самолюбованию. Мне просто хочется, чтобы вы осознали некоторые особенности роли лидера и попытались правильно ими воспользоваться. Всем содержанием главы о совещаниях я хотел побудить вас сфокусироваться на отношениях с окружающими.

Консенсус и действия в результате совещаний

В этой главе я упомянул о нескольких типах совещаний. Все они характеризуются единой целью, которая состоит, во-первых, в достижении консенсуса среди участников, и, во-вторых, в составлении плана дальнейших действий. Эти задачи решаются за счет профессионального поведения на любых совещаниях, вне зависимости от того, насколько формальный или неформальный характер они носят. Что я имею в виду под профессиональным поведением? Согласно Вебстеру (Webster), одним из аспектов профессионализма является «последовательное, методичное проведение каких-либо действий». Так и в нашем случае. Последовательность в деле проведения совещаний в конечном итоге приведет вас к успеху. Дискуссии на совещаниях следует поощрять, поскольку на их основе рождаются идеи. Ваша компетенция — еще один аспект профессионализма — в ходе дискуссий, призванных перетекать в конкретные действия, должна быть очевидной.

Попробуем обобщить все вышесказанное. Итак, участвуя в совещаниях или ведя совещания, придерживайтесь следующих принципов.

- Не превращайте совещание в партсобрание. Участие в совещаниях — это тоже работа, и цель состоит в том, чтобы приступить к новой рабочей неделе осмысленно.
- При проведении проектных совещаний обязательно подготавливайте четкий план действий. Руководите разумно и учитывайте фактор группового поведения.
- Проводя беседы с сотрудниками один на один, следуйте принципам, характерным для совещаний в более крупном составе: пытайтесь достичь консенсуса и выстроить план действий.
- Проводя совещания с нетехническими специалистами, помните, что вы олицетворяете образ технаря. Пытайтесь доносить сложные идеи простым языком. Попытки произвести на бухгалтерских работников впечатление, жонглируя таинственными сокращениями, вряд ли прибавят вам уважения. На таких

совещаниях ведите себя как педагог. В то же время исполняйте эту роль предусмотрительно, избегайте высокомерия.

- Проводя ретроспективные совещания по проекту, не поддавайтесь соблазну свалить все грехи на другие отделы. Попытайтесь выяснить, в каких областях вы сами и ваши сотрудники могли бы добиться более высокой продуктивности.
- Старайтесь свести количество телеконференций к минимуму. Путем тщательной подготовки и составления четкого плана старайтесь делать их как можно насыщеннее.

Что дальше

Многое из того, что я рассказываю в этой книге, носит личностный оттенок. Действительно, трудно придумать тему более личностную, чем лидерство при проведении совещаний. В следующей главе я расскажу, как исполнять роль технического лидера. Здесь вам также потребуются навыки группового общения, хотя, располагая обширным опытом взаимодействия с технологиями (по крайней мере, большим, чем общения с людьми), вы, скорее всего, будете чувствовать себя в этой роли более комфортно. Поймите меня правильно — я даже мысли не допускаю, что всю свою жизнь вы прожили в изоляции от общества. Просто технические навыки составляют основу вашего участия в общей деятельности. Не бросайте чтение — следующая глава вам, скорее всего, понравится больше, чем все предыдущие.

Глава 6

Философия и методы технического лидера

Весьма высока вероятность того, что причина, по которой вас сделали руководителем, заключается в ваших успехах на поприще техники. Это свое предположение я уже ранее высказывал. И хотя технические навыки далеко не всегда гарантируют качество руководства, для исполнения роли, которую я намерен описать в этой главе, вы подходите идеально. Многое из того, о чем я рассказывал в предыдущих главах, полагаю, привело вас в уныние своей новизной; материал же этой главы, напротив, скорее всего, покажется вам знакомым и, может быть, даже тривиальным. Это совершенно не означает, что читать главу не надо, — не зря же я ее написал, в конце концов! На самом деле взглянуть на технические проблемы с точки зрения технического лидера очень полезно. Ведь теперь вы не просто «технический руководитель» группы; вы — лидер с большой буквы. Принятие решений — это ваша прерогатива. За последствия принятых решений, сказывающихся на разрабатываемых программных продуктах, вы также несете ответственность, поэтому проблема выбора и принятия решений выходит на первый план.

Каким образом в процессе принятия решения вам поможет эта глава? В ней мы рассмотрим некоторые технические принципы и порассуждаем о деталях — тем самым я попытаюсь привить вам уверенность в своих действиях. С моей точки зрения, главное в нашей высокотехнологичной профессии — сделать так, чтобы лидерство осуществлялось на основе некоего философского видения, раскрывающего технические навыки ведущих программистов. Этой философской основой мы займемся в первую очередь. Не стоит, правда, забывать, что без принципов ее практической реализации полноценное лидерство невозможно, и, соответственно, успешно пасти котов вам также не удастся.

Как уразуметь свою техническую роль и придерживаться ее

Роль технического лидера заключается в том, чтобы координировать архитектурные и проектные решения. Архитектура и проектирование — это, как известно, две разные вещи. Да, действительно, программисты иногда воспринимают их как синонимы, но, я вас уверяю, это в корне неправильно. Практикуя техническое лидерство в масштабе своего отдела, вы скоро убедитесь в том, что проектирование компонентов приложений очень сильно отличается от разработки их общей архитектуры. Архитектура направлена на многократное использование проектных решений. В идеале перед сборкой системы из составляющих ее компонентов неплохо было бы макетировать архитектурное понятие. В этом контексте вспомните компонентное конструирование из области объектно-ориентированной технологии — своей основной целью оно провозглашает многократное использование кода. Итак, мы имеем дело с двумя совершенно разными задачами, и по моему скромному, но правильному мнению, многократное использование проектных решений значительно важнее.

Архитектура ориентирована на многократное использование проектных решений.

Сила объектно-ориентированной технологии вне зависимости от языка, на котором она реализуется, — это возможность создания объектов, связывающих прикладные данные с вариантами пользовательского взаимодействия. Детали при этом скрываются. Таким образом, избегая непосредственного соприкосновения с уровнями данных и пользовательского интерфейса, объекты получают открытые интерфейсы, которые, в свою очередь, взаимодействуют со структурой программы и процессами. Итак, мы получаем возможность собирать компоненты, согласующиеся со свойствами и методами, имена которых значимы в контексте проблемной области (в противоположность области решений). К примеру, метод под именем `ShowAppointments`, наполняя свойство `Appointments` графического пользовательского интерфейса, способен одновременно скрывать детали — такие как `Select * from Appointments where Date = Today`. Таким образом, удобочитаемость программ открывает возможности для решения конкретных задач.

Именно в этом высоком уровне понимания кроются огромные преимущества объектно-ориентированных методов, за счет которых последние выгодно отличаются от функциональной декомпозиции и структурных методик, доминировавших в более ранний период компьютерного программирования. В то же время узкое место объектно-ориентированной технологии приходится как раз на создание общей архитектуры системы. С одной стороны, объектно-ориентированные методики позволяют успешно проводить архитектурный анализ; с другой — навыки,

полученные при создании компонента, не соответствуют напрямую тем навыкам, которые требуются для организации целостной системы компонентов. Это белое пятно в наборе средств объектно-ориентированной технологии как раз закрывается архитектурными методиками. Ваша задача как технического руководителя состоит в том, чтобы обеспечить создание системной архитектуры до принятия решений по технологии реализации и деталям компонентов, из которых данную систему предполагается конструировать.

Обратимся к аналогии. Классическую архитектурную традицию западной цивилизации принято отсчитывать от древнегреческих и древнеримских строений. Эти здания, простоявшие уже более двух тысяч лет, подобно литературным произведениям талантливых авторов, доносят до непосвященного наблюдателя простое и ясное послание. В то же время при более детальном анализе структуры этих зданий выясняется, что в них скрыты многочисленные свидетельства дотошной творческой работы. Программная архитектура способна на такие же чудеса — демонстрируя внешний блеск, она может заключать в себе невероятную сложность. Впрочем, поскольку наша дисциплина существует всего лишь полвека, до того момента как ее лучшие достижения станут называть классическими, пройдет еще много времени. Как технический лидер вы должны заложить те основы, на которых сотрудникам вашей группы и, возможно, другим разработчикам предстоит заниматься непосредственно конструированием. Да, мы живем в то время, когда популярное искусство необдуманно провозглашается непреходящей ценностью. При этом не следует забывать, что результаты наших трудов над клавиатурой должны пройти проверку временем.

Конструировать или выращивать

Традиционной метафоре строительства в нашей профессии соответствует термин «конструирование»[1]. Как часто мы им пользуемся для описания своих действий? По моим наблюдениям, довольно часто. Все начинается с детства. Если вы из моего поколения, то ребенком, наверное, играли с конструкторами — ну а представители поколения X, вероятно, помнят свои экзерсисы с Lego. Обиходное понятие «конструирование программного обеспечения» вполне доходчиво обозначает повседневную деятельность программиста. Попробуем проанализировать эту метафору в буквальном смысле и объяснить, что значит дополнять приложения новыми характеристиками. Можно ли надстроить небоскреб дополнительными десятью этажами и при этом пребывать в уверенности, что фундамент их выдержит? Мне так не кажется; надеюсь, что и вы придерживаетесь моего мнения. Придется обратиться к еще одной метафоре — садоводству. Разбивая сад и выращивая его, вы, естественно, время от времени выкорчевываете одни растения и высаживаете другие. В своей книге о прагматическом программировании Эндрю Хант (Andrew Hunt) и Дэвид Томас (David Thomas) высказываются об этой метафоре следующим образом:

> «Растения в саду высаживаются, с одной стороны, в соответствии с исходным замыслом, а с другой — согласно текущим обстоятельствам. Некоторые из них

[1] И строительство, и конструирование выражаются в английском языке одним словом — construction. Поэтому нам при переводе пришлось в меру сил лавировать между этими двумя значениями. Таким вот печальным образом гипертрофированная метафоричность автора разбивается об языковой барьер. — *Примеч. перев.*

выживают, другим же суждено превратиться в компост. Растения можно пересаживать, менять их расположение, играя, таким образом, со светом и тенью, ветром и дождем. Переросшие растения подрезают или срезают, а если, к примеру, какой-нибудь цветок по своему цвету не соответствует окружению, его пересаживают в более подходящее (с эстетической точки зрения) место. Занимаясь садоводством, мы выдираем сорняки и удобряем растения, которым нужна дополнительная поддержка. Хороший садовод постоянно наблюдает за здоровьем растений на своем участке и при необходимости вносит разного рода коррективы (удобряет почву, пересаживает растения, придумывает новый вариант разбивки)»[1].

Надо сказать, что метафора садоводства значительно лучше соответствует разработке программных средств. Она делает очевидным преходящий характер программных продуктов и акцентирует внимание на архитектуре, гибкость которой подразумевает возможность реализации новых структур. В контексте архитектуры системы конструируются, а непосредственно программные продукты выращиваются. Чтобы соответствовать потребностям бизнеса XXI века, мы должны построить органичную методику разработки программных продуктов.

Чтобы соответствовать потребностям бизнеса XXI века, мы должны построить органичную методику разработки программных продуктов.

Метафора садоводства, к тому же, выражает различие между органическим и синтетическим. Все органическое выращивается; синтетические же объекты собираются из конструируемых компонентов. Действительно, конструируя компоненты, мы делаем их синтетическими по самой природе. Однако та среда, в которой их предполагается выращивать, должна выводить целое за рамки суммы компонентов. Кроме того, она должна предусматривать возможность адаптации к изменяющимся коммерческим требованиям и к технологической эволюции.

Итак, пытаясь усвоить материал, который я излагаю в дальнейшем, не забывайте о биологии и старайтесь не зацикливаться на аналогиях со строительной индустрией. Функции метафор ограничены — они просто-напросто помогают применить абстрактные понятия в условиях рутинной технической деятельности. Нельзя писать код, исходя из метафоры. В этом процессе без деталей проектного решения не обойтись. С другой стороны, все эти детали в совокупности формируют образец. А вот образец уже можно рассматривать метафорически, анализируя степень его применимости к решению бизнес-задач. Вы ведь еще не забыли, как важно для нас думать? Штудируя мою книгу, вы должны были уже уяснить, что главное в нашей работе — мыслить. Мыслить широко и ни в коем случае не поверхностно. Никогда не забывайте этот принцип — он вам пригодится.

Примат архитектуры

За последнее десятилетие появилось множество работ — как фундаментальных, так и прикладных, — доказывающих необходимость применения в процессе раз-

[1] Andrew Hunt and David Thomas, The Pragmatic Programmer (New York: Addison-Wesley, 2000), p. 184.

работки программных средств качественной архитектуры. Исследователи, специализирующиеся в этой области, указывают на то, что огромное количество проектов приложений заканчиваются неудачей именно из-за пренебрежения архитектурными вопросами[1]. В результате выполнения таких проектов, как правило, получаются «прямолинейные» системы. Их последующая адаптация к изменяющимся коммерческим требованиям осуществляется с большими трудностями, поскольку предполагает выделение значительных временных, трудовых и интеллектуальных ресурсов.

Занимаясь проектированием той или иной системы, вы должны уделять первоочередное внимание рискам. Каким видам рисков? Во-первых, риску ухудшения позиций компании на рынке, возникающему, когда в существующий продукт вследствие конкуренции приходится вносить новые непредусмотренные изначально черты; риску неумеренного усложнения процесса сопровождения продуктов, возникающему вследствие жесткой взаимозависимости компонентов-подсистем или негибкости их конфигурации. Еще один вид рисков заключается в неоправданной сложности продукта на верхних уровнях архитектуры. Это обстоятельство чрезвычайно усложняет задачу кодировщиков, не принимавших участия в процессе создания системы, но вынужденных выискивать способы исправления базовых компонентов. Все перечисленные проблемы имеют непосредственное отношение к временным и финансовым затратам, которые ваш начальник, естественно, желает сократить.

Создание архитектуры — это активный творческий процесс, который отнюдь не ограничивается сидением за клавиатурой, фиксацией коммерческих требований и реализацией компонентов, которые способны эти требования удовлетворить, в коде. В процессе работы над архитектурой вам придется абстрагироваться от своих механистических обязанностей и максимально углубиться в задачу, которую предстоит решить. Спору нет — во многих случаях решение оказывается вполне механистическим, то есть чисто программным. И тем не менее, если вы не разберетесь в задачах, стоящих перед своей компанией, обеспечить продуктам длительное и продуктивное существование вам, скорее всего, не удастся. Марк и Лора Сьюелл (Marc & Laura Sewell) в своей работе о роли архитекторов перечисляют ряд важнейших действий, которые должны предшествовать составлению любого проектного плана[2]. С точки зрения этих авторов, любой архитектор должен:

- в совершенстве владеть искусством выслушивания, опрашивания и наблюдения;
- хорошо разбираться в предметной области клиента — будь то банковское обслуживание, деятельность государственных органов, образование, здравоохранение, розничная торговля или скачки;
- получить стратегическое представление о деятельности предприятия клиента, не ограничиваясь тактическим или рабочим обзором его деятельности;

---

[1] Рекомендую в этом контексте ознакомиться с работами апологетов позитивных и негативных эталонов, например с трудами Брауна (Brown) и других исследователей, чьи публикации перечислены в библиографии.

[2] Marc T. Sewell and Laura M. Sewell, The Software Architect's Profession (Upper Saddle River, NJ: Prentice Hall, 2002), p. 68.

- иметь комплексные познания в технологической области — для того чтобы при разработке архитектурного плана суметь учесть все без исключения варианты стратегических решений;
- наладить конструктивное взаимодействие с клиентом и специалистом, ответственным за конструирование;
- уяснить видение вопроса клиентом и согласованное с ним проектное решение, следить за их неукоснительным соблюдением.

Не кажется ли вам, что выполнение всех этих требований выходит за рамки ваших скромных обязанностей? Если вы придерживаетесь такой точки зрения, рекомендую вам расширить кругозор, предварительно попытавшись вспомнить все разработанные с вашим участием продукты, которым так и не суждено было продвинуться дальше первой версии. Все мы так или иначе имели дело с продуктами-однодневками. С моей точки зрения, недостаточно комплексное понимание задачи приводит к созданию подобного рода тактических решений, которые на первый взгляд удачны, однако не выдерживают проверки временем.

Архитектор должен уметь решать эту проблему, имея в виду два важных понятия: *проектные ограничения* (design forces), которые обусловливают все решения, принимаемые относительно программных средств, и *аналитические позиции* (analysis viewpoints), позволяющие принимать решения в соответствии с проектными ограничениями.

Проектные ограничения в архитектурном планировании

Поскольку в формализованном виде дисциплина архитектурного планирования появилась относительно недавно, в ее рамках существует несколько конкурирующих концепций. Мальво (Malveau) и Маубрей (Mowbray)[1] — два эксперта в этой области — приводят список основных концепций. Это каркас Закмана (Zachman), открытая распределенная обработка (Open Distributed Processing, ODP), анализ предметной области, модель 4+1 представлений программной архитектуры и, наконец, академическая программная архитектура. Это уже немало. А знакомы ли вы с положениями каждой из них? Если нет, торопитесь — принимайтесь за изучение перечисленных концепций, поскольку они предоставляют любому специалисту прекрасный инструментарий для создания многократно используемых систем.

Разобравшись с положениями различных архитектурных концепций, вы поймете, что все они преследуют общую цель. Она заключается в контроле над проектными ограничениями, которые обуславловают все без исключения программные решения, относящиеся к архитектуре. Проектные ограничения — это те факторы, которые необходимо учитывать с самого начала процесса разработки программного продукта. Они выражают ряд вопросов, на которые любая архитектура должна отвечать. Частично эти вопросы перечислены ниже.

- Сможет ли система предоставить пользователю комфортные условия работы и функционировать согласно его ожиданиям?

---

[1] Raphael C. Malveau and Thomas Mowbray, Software Architect Bootcamp (Upper Saddle River, NJ: Prentice Hall, 2001).

- Может ли система функционировать в соответствии с проектным решением, предусматривает ли она модификацию и сопровождение по мере изменения коммерческих потребностей и технологии?
- Минимизирована ли сложность высокоуровневых архитектурных характеристик системы?
- Какой бы продукт ни создавался, в течение нескольких последующих лет его ожидают значительные изменения, причем предпосылки для осуществления этих изменений должны закладываться в фундамент проекта.
- В связи с тем, что ожидается реализация решений в аппаратной части, особое внимание следует уделить созданию, конфигурированию и сопровождению инфраструктуры.
- Достаточна ли компетентность персонала для сопровождения систем, построенных на основе новых технологий? Если в конструировании[1] применяются старые технологии, насколько они перспективны с точки зрения продолжительности использования системы?

Ни в коем случае не забывайте об этих вопросах и проблемах. Обязательно сформулируйте их и донесите до группы разработчиков — ведь умение четко сформулировать вопросы иногда важнее, чем даже знание правильных ответов на них. Может быть, конечно, в этом я преувеличиваю, но, полагаю, мысль вам понятна. Способов наделать глупостей всегда более чем достаточно, и первый вопрос, который вы должны себе задать, звучит так: «Что я должен делать? Какое решение будет правильным?» Задавая адекватные вопросы, вы сможете организовать создание стройной, мощной и долговечной архитектуры.

Аналитические позиции как средство управления проектными ограничениями

Чтобы решить проблемы, возникающие благодаря проектным ограничениям, необходимо иметь представление о позициях, с которых анализируются любые корпоративные варианты архитектуры. Аналитическая позиция — это, по сути, точка зрения на проектное решение, исходя из которой оно анализируется. Эта позиция, или точка зрения, изменяется по мере изучения системы с разных сторон, исходя из степени серьезности различных проектных ограничений. Вне зависимости от конкретной концепции создания архитектуры все исследователи выделяют несколько общих аналитических позиций, формулируя их в виде вопросов.

- Как будет проходить взаимодействие пользователей с системой? (Эту аналитическую позицию часто называют вариантной.)
- Какие компоненты требуется собрать для того, чтобы обеспечить функционирование системы?
- Каков механизм взаимодействия компонентов, благодаря которому система функционирует?

---

[1] Как просто, оказывается, вернуться к привычной терминологии! Если бы я сказал «...в выращивании применяются старые технологии...», смогли бы вы понять, о чем я говорю?

- Какие технологии в наибольшей степени приспособлены для создания данного программного обеспечения?
- Как предполагается поставить систему клиенту?

Задаетесь ли вы этими вопросами о предполагаемом продукте в ходе проектных совещаний (о них мы говорили в предыдущей главе)? Без них не обойтись — в противном случае у вас получится случайная архитектура, а это крайне опасный негативный эталон. Не забывайте, что переработать архитектуру значительно сложнее и потенциально разрушительнее, чем реконструировать компоненты. Этот фактор риска в процессе проектирования следует постоянно иметь в виду.

Переработать архитектуру значительно сложнее и потенциально разрушительнее, чем реконструировать компоненты.

Держу пари, что вы сомневаетесь: нужно ли вам все это знать? Полагаю, что если вы действительно хотите стать эффективным техническим лидером, это знание необходимо. Если вы пользуетесь языком программирования четвертого поколения (4GL, например, Visual Basic), а не, скажем, C++, то больше внимания следует уделять разработке архитектуры системы. Даже при условии применения C++ создать плохую архитектуру не представляет труда — что уж говорить о языках четвертого поколения! Да, действительно, они позволяют оперативно выстроить механизм взаимодействия пользователя с системой, но это, как вы понимаете, лишь ее оболочка. Термин *быстрая разработка приложений* (Rapid Application Development, RAD), возникнув на заре создания программных продуктов под Windows, первоначально отражал высокие темпы выхода продуктов на рынок, достигавшиеся средствами языков четвертого поколения. В сегодняшних условиях «быстрая» разработка в большинстве случаев приводит к появлению некачественных программных продуктов. Мне даже кажется, что аббревиатуру RAD сегодня более уместно расшифровывать как «разработка мерзких приложений» (Rotten Application Development). Первоочередное внимание следует уделять, говоря анатомическим языком, скелету, нервной системе и внутренним органам. В противном случае вы рискуете извергнуть очередную халтуру.

Некоторые компании настолько увлекаются скоростью разработки, что рано или поздно им придется столкнуться с долгосрочными последствиями непродуманности архитектуры. Кто знает, быть может, вы работаете в одной из таких компаний, трудитесь себе над продуктом, находящимся в конце цикла разработки, и регулярно выслушиваете от сотрудников группы поддержки отчеты о найденных ошибках. Стремление ввести в продукт новые характеристики естественным образом ограничивается необходимостью вернуться к исправлению найденных ошибок, которые, вполне возможно, возникли именно из-за того, что в процессе разработки был взят слишком высокий темп. Если бы у вас нашлось время поразмыслить над этой проблемой, вы, скорее всего, пришли бы к выводу о том, что быстрая работа существенно увеличивает риск в ближайшее же время столкнуться с неприятностями, обусловленными непродуманностью архитектуры. Процесс конструирования программных продуктов, если обратиться на этот раз к спортивной

терминологии, можно сравнить с марафоном, но уж никак не со спринтом. Темп, предусматривающий долговечность конечного результата, есть необходимое условие достижения качества в производстве.

Процесс конструирования программных продуктов можно сравнить с марафоном, но уж никак не со спринтом. Темп, предусматривающий долговечность конечного результата, есть необходимое условие достижения качества в производстве.

Как же, отказавшись от синтетического подхода в пользу органического и от быстрой разработки с неудовлетворительным результатом, создать стройную архитектуру? Мыслить органично вам поможет свежий взгляд на принципы проектирования и очередность этапов фазы разработки.

Свежий взгляд на проектирование

Как отличить проектное решение от архитектуры? Представьте себя божком, собирающимся сотворить живое и разумное существо, обладающее, к тому же, способностью к адаптации. Надеюсь, для вас это не проблема (кстати, если так, придется вам внимательно ознакомиться со следующей главой, посвященной темной стороне лидерства). Как бы там ни было, трудно сотворить часть тела, не понимая, в каком окружении она будет существовать. К примеру, если бы легкие висели на левой руке, вряд ли они смогли бы исполнять свою основную функцию, — значит, лучше разместить их в груди. Полагаю, вы понимаете, к чему я клоню. При создании проектного решения предполагается наличие архитектуры, которая диктует расположение всех реализующих системные функции компонентов. Таким образом, проектное решение становится «плотью»[1] архитектуры; кроме того, на этом этапе разработки производится выбор технологии реализации.

О чем вы говорите? Я предпочитаю VB и собираюсь писать на нем все программы[2]. Вы так считаете? Подумайте еще раз. Задачи, которые вам предстоит решать, — не гвозди; их нельзя вбивать одним молотком. Да, я опять обращаюсь к метафоре строительства, но, по-моему, в этом контексте она как нельзя более кстати. Не забывайте, впрочем, что метафоры и аналогии выполняют исключительно иллюстративную функцию — они как луч света в темной комнате, освещающий очертания меблировки. Иначе говоря, иллюстрация и реальный объект не идентичны; между иллюстрацией и проблемной областью нет точного соответствия. Аналогичным образом проектные решения можно принимать только при наличии утвержденной архитектуры.

В последующих разделах мы поговорим о том, как довести грамотно выращенный продукт до стадии сбора урожая. Вопрос заключается в следующем: является ли этот процесс многоступенчатым?

---

[1] Ах, какой я молодец, что подобрал анатомическую метафору!

[2] Лично я пользуюсь VB, начиная с версии 1.0, поэтому никаких предрассудков против этого языка не питаю. Всем интересующимся проблемами создания архитектуры и проектирования средствами VB рекомендую ознакомиться с работой Billy S. Hollis, Visual Basic 6 Design, Specification and Objects (Upper Saddle River, NJ: Prentice Hall, 1999).

Нулевой этап проектирования

Итак, имея на руках грандиозную архитектуру, вы готовы приступить к проектированию компонентов. Прекрасно. Мобилизуйте все свои объектно-ориентированные навыки. В первую очередь вам предстоит проверить архитектуру, имея в виду подчеркнуть приоритет этого этапа (я называю его «нулевым») в контексте процесса проектирования. Другими словами, вы должны провести макетирование архитектуры, но не совсем так, как это делается обычно. Вместо обширного графического пользовательского интерфейса сконструируйте ряд низкоуровневых компонентов, снабдите их интерфейсами-заглушками и возвращаемыми значениями — это позволит убедиться в работоспособности системы в вертикальной проекции. Конструируемые на этом этапе графические пользовательские интерфейсы должны лишь подавать сигналы и запускать те или иные процессы — больше от них ничего не требуется. Ведь ни один человек не выживет, если его мозг не будет взаимодействовать с сердцем, так? Именно поэтому, кстати, косметические операции проводятся исключительно по желанию пациента, а вот при проведении операции на головном мозге его согласия не требуют. Я полностью убежден — проверять архитектуру необходимо. На это, скорее всего, уйдет больше времени, чем можно было бы ожидать, но не сомневайтесь — усилия стоят того. С соблазном сразу перейти к разработке реальных компонентов системы нужно бороться — будет очень неприятно, если, подключив все компоненты, вы обнаружите, что они не стыкуются или не работают.

Первое, что нужно сделать в процессе проектирования, — это проверить архитектуру.

Подробности процесса «проверки концепции» я, пожалуй, опущу — благо литературы, посвященной архитектуре и проектированию, предостаточно. Только вот не стоит перекладывать обязанности по проверке архитектуры на кого-то другого — и не дай вам Боже отложить этот процесс до бета-тестирования. Эта обязанность ложится на вас и ваших подчиненных, а основная идея, которую я стремлюсь до вас донести, ясна и понятна — нельзя уклоняться от выполнения своих обязанностей. Ваша первоочередная задача в процессе проектирования заключается как раз в том, чтобы проверить архитектуру на предмет ее применимости — вот почему речь идет о «нулевом», то есть о приоритетном, этапе. Если на все нижеследующие вопросы вам удастся ответить положительно, значит, вы *наверняка* на правильном пути.

- Предполагает ли архитектура возможность идентификации подсистем функций и предотвращает ли она их взаимозависимость с другими подсистемами? Удалось ли вам сконструировать объекты, способные функционировать сравнительно независимо при помощи ограниченного числа вспомогательных объектов?
- Насколько доходчиво «перспективное представление» системы — сможет ли, скажем, продавец, прослушав краткий инструктаж, объяснить потенциальным

клиентам, как работает программный продукт, который они намереваются приобрести?

- Исключается ли жесткое кодирование параметров конфигурации системы? К примеру, потребуется ли повторная компиляция программы при переименовании сервера или домена, или редактирования конфигурационного файла будет достаточно?

Обратите внимание на слово «наверняка», выделенное курсивом в последнем предложении абзаца, предшествующего списку. Почему я употребил именно это слово? Дело в том, что вы должны учитывать все факторы воздействия на систему на 1–2 года вперед. Невозможно выстроить надежный план действий, не зная коммерческой конъюнктуры. С другой стороны, разбираясь в тонкостях бизнеса, вы имеете все шансы стать предсказателем будущего своих программных продуктов.

Почему я говорю «предсказатель», а не, скажем, «прогнозист»? Объясняется это следующим образом. Термин «предсказатель» (prognosticator) образуется из двух греческих корней: *pro*, что в переводе означает «до», и *gnosis*, то есть «знание». Нельзя точно знать, что произойдет в будущем, однако, как и специалисты, составляющие прогнозы погоды, чем больше мы практикуемся по части предсказаний, тем лучшие результаты получаем. Не стоит также забывать, что создание программных продуктов — это искусство, которое постепенно приобретает очертания науки. Любая наука начинается с исследования явлений (феноменологии). В контексте разработки программных средств эта деятельность сводится к документированию всех явлений, наблюдаемых в ходе процесса, и построению эталонов. По мере исполнения этой миссии начинают выкристаллизовываться закономерности, согласно которым у нас что-то получается или, наоборот, не получается. В конце концов нам удается сформулировать принципы кодирования. Когда кому-нибудь это удается, он присваивает новому принципу свое имя, публикует книгу и наслаждается признанием. Именно этим в последнее десятилетие занимались разработчики концепции позитивных (pattern) и негативных (antipattern) эталонов, в связи с чем ваши шансы на славу теперь минимальны.

На самом деле, если вам удастся найти в цепочке производства прибыльных программных продуктов слабое звено, быть может, вы и не получите международного признания, но рост престижа среди сотрудников собственной компании вам обеспечен. Не исключено также, что ваша компания не испытывает подобных проблем. Что ж, тем больше перед вами открывается возможностей. Впрочем, большинство представителей нашей профессии еще только на пути к программной нирване и поэтому без посторонней помощи обойтись не могут. И у вас как у технического лидера есть все возможности помогать окружающим.

Этапы проектирования 1, 2, 3, 2, 1, 4...

Итак, вы готовы запустить маховик проектирования. Теперь дела должны пойти в гору. То есть, я имею в виду, с горы. Да, я полон противоречий. Но ведь и процесс проектирования, как и жизнь, похож на американские горки. Попытайтесь получить удовольствие от поездки, смакуйте азарт и не забудьте вернуться живым. Мне кажется, процесс проектирования в его лучшем понимании правомерно сравнить с катанием на винтообразной (в противоположность круговой) трассе. В ней есть начало и конец, но в то же время вашему взору регулярно открываются одни и те же виды. А теперь сравните это удовольствие с падением с водопада, когда

вас несет вниз по реке, потом вы оказываетесь в свободном полете и, наконец, шлепаетесь пятой точкой об воду, рассчитывая при этом остаться в живых. Такое впечатление, что вы работаете в парке аттракционов, не так ли?

Нет, тут я неправ; слово «развлечение» (amusement), опять же, образуется из двух греческих составляющих: *a*, что означает «нет», и *muse* в значении «думать». Я, равно как и ваш начальник, все-таки надеюсь, что в процессе проектирования вы думаете. В данный момент я пытаюсь доходчиво объяснить различия между устаревшим водопадным циклом разработки и разрекламированным в последнее время итеративным циклом. Я убежденный сторонник последнего — несмотря даже на то, что иногда он кажется бесконечным. Как бы там ни было, если вы стремитесь к эффективному проектированию, необходимо неизменно придерживаться архитектурного плана. Существует множество книг, проводится множество семинаров, на которых нас учат «правильным» методам проектирования. Некоторые из них действительно очень полезны. Лучшим же методом проектирования мне представляется тот, которым вы и ваши подчиненные уже привыкли пользоваться, и заключается он в решении корпоративных задач. Если вы еще не достигли в этом отношении больших успехов, не стоит себя корить. Наша работа не обходится без трудностей. Некоторые из нас совершенствуются, иные же через пару лет просто исчезают из поля зрения. Это жестокий мир, так что крепитесь и стремитесь к совершенствованию своих методов.

В предыдущем абзаце я обратился к метафоре из Дарвина по поводу того, что выживает сильнейший. Родившийся в XIX веке, этот замечательный принцип и поныне применяется при оценке корпоративной культуры и деятельности сотрудников предприятий. Во многих случаях он вполне адекватен; не будем, впрочем, забывать о других концепциях из области эволюционной биологии, в частности об адаптации. В настоящее время зарождается новый подход к адаптации. Предлагают его те исследователи, которые занимаются вопросами поведения сложных систем и принципами самоорганизации в таких системах. Стюарт Кауфман (Stuart Kauffman), один из ведущих специалистов в этой области, утверждает, что «...как биологическая, так и технологическая эволюция представляет собой процессы, направленный на оптимизацию систем с конфликтующими ограничениями»[1]. Для более осмысленного освещения вопросов выживания организмов и их существования в хаосе сложных систем Кауфман предлагает выражение «поступление сильнейших». Вооружившись его видением проблемы, исследователи привязали его выводы к процессы разработки программных продуктов.

Прекрасный образец применения теории сложных систем в области разработки программного обеспечения являет собой книга Джеймса Хайсмита (James Highsmith) под названием «Адаптивная разработка программных средств». Хайсмит предлагает вниманию читателя итеративный цикл, который он называет «жизненным циклом адаптивной разработки». В нем три основных компонента: сотрудничество, размышление и обучение. С его точки зрения, у этого цикла есть ряд преимуществ[2]:

- реагируя на периодические отзывы, приложения эволюционируют, приближаясь тем самым к предъявляемым заказчиками требованиям;

---

[1] Stuart Kauffman, At Home in the Universe (New York: Oxford University Press, 1995), p. 179.

[2] James A. Highsmith III, Adaptive Software Development (New York: Dorset House Publishing, 2000), p. 40.

- упрощается адаптация к изменяющимся бизнес-требованиям;
- процесс разработки подгоняется под заданный для данного продукта профиль качества;
- преимущества от применения продукта заказчик получает раньше, чем обычно, — хотя бы за счет того, что приложение поставляется ему быстрее, а значит, он раньше начинает получать доход;
- заказчики раньше, чем обычно, начинают испытывать доверие к проекту.

Какое отношение все это имеет к проектированию? В любом случае в вашей компании применяется какой-то метод проектирования — он либо адекватен, либо требует усовершенствования, либо не годится для решения поставленных задач. Возможно, требуется его немедленная замена. Решение за вами. Я считаю, вы должны проявить инициативу, проанализировать применяемые в компании методы проектирования и подогнать их под заведомо работоспособные образцы.

Теперь пора спуститься с теоретических высот (а теория эта весьма достойна) и обратиться в следующем разделе к практическим методам разработки программных продуктов, которые, по моему опыту, дают хороший результат. Мне кажется, что они носят довольно общий характер и, следовательно, могут применяться любой группой разработчиков вне зависимости от используемого ими языка программирования.

Принципы проектирования

Коль скоро мы придерживаемся принципа органической архитектуры, нам нужны органические компоненты. Как появляется программный объект? Естественно, в результате написания кода — как это ни прискорбно, если мы нашепчем компьютеру через микрофон идею объекта, он не появится. Собственно говоря, в такой идее нет ничего плохого — в особенности если в ней заключены принципы кодирования, подобные следующим.

- Следование стандартам программирования, принятым для данного языка[1]. Это обеспечивает единообразие методик конструирования объектов, применяемых разными программистами. (В этом контексте имеет полное право на существование метафора строительства.)
- Поощрение связности объектов. Не следует воспринимать объекты как контейнеры для размещения совокупности процедур — скорее это органы, выполняющие конкретную функцию. Как известно, сердце не пытается дышать, а легкие не качают кровь.
- Ограничение взаимозависимости объектов. В отсутствие серьезных аргументов в пользу иной точки зрения взаимозависимость приносит одни неприятности, превращая сопровождение в сплошной кошмар[2]. Чтобы однажды сде-

---

[1] «Стандарты программирования» — выражение многозначное. Диапазон его значений простирается от инструкции по написанию качественной процедуры до утверждения единственной точки выхода из подпрограммы. Список этот можно продолжать бесконечно. Объединяющий принцип прост — заведите стандарт и придерживайтесь его.

[2] Что такое кошмар для программиста? Это когда он вынужден не спать всю ночь, исправляя код, в котором, внеся одно исправление в одном объекте, получает изменение в трех местах другого объекта.

ланные взаимозависимыми объекты превратить в независимые, требуются дополнительные временные и финансовые ресурсы.

Со временем, по мере того как вы будете нарабатывать опыт руководства проектами, приведенный список можно будет расширить. Наилучшие методики деятельности в области разработки программных средств обнаруживаются и утверждаются постепенно. Не сомневаюсь в том, что у вас есть несколько любимых авторов, чьи труды по мере обучения серьезно помогли. Если так, не изменяйте им. На тот случай, если в вашей личной библиотеке не хватает полезных книг, я составил библиографический список. Ведь учиться у предшественников и современников совершенно необходимо. Сегодня в качестве руководства вы выбрали мою книгу. Я стремлюсь к тому, чтобы поведать вам суть различных принципов разработки и объяснить причины, по которым вы как технический лидер должны пропагандировать эти принципы среди своих подчиненных.

Принципы успеха

Даже самая шикарная библиотека не гарантирует успешной деятельности в роли технического лидера. Ее наличие — это лишь одно из многочисленных условий достижения профессиональных высот. Ничто не мешает вам выстроить на полках увесистые тома, пытаясь тем самым впечатлить себя и окружающих. Но от этого вы не станете лучше как технический руководитель. Обширная библиотека должна быть в ваших мозгах — лишь при таком условии вы сможете взвешенно организовать процесс проектирования. Взвешенность приходит с опытом, являясь следствием осмысления ошибок. Вот почему каждый раз, когда я совершал какую-нибудь ошибку, мой отец говорил «мотай на ус». Конечно, некоторые ошибки не проходили бесследно, однако понимание того, что это в порядке вещей, меня несколько успокаивало.

Обширная библиотека должна быть в ваших мозгах — лишь при таком условии вы сможете взвешенно организовать процесс проектирования.

А вы боитесь совершать ошибки? Не стоит питать иллюзий — ошибок в суждениях в процессе руководства подчиненными вам не избежать. Ошибки допускаются регулярно, и иногда за них приходится отвечать по полной программе — в особенности когда вы не успеваете к контрольным срокам. Впрочем, с опытом вы усовершенствуете свои навыки, и, будем надеяться, проблемы с соблюдением сроков отпадут — ох, как же вас тогда зауважают сотрудники других отделов! На входе в офис вас будут встречать восторженные поклонники и петь вам гимны. Ну, может быть, я немножко преувеличиваю, но в одном нисколько не сомневаюсь — если вам удастся повысить продуктивность сотрудников своей рабочей группы, ваши уверенность в себе и преданность работе обязательно поднимутся до заоблачных высот. Одержимость работой — вещь заразная, и вы должны всеми силами стараться распространить ее на всех своих коллег.

Теперь вернемся к конкретике: есть ли у меня какое-то универсальное решение? Есть, и я о нем уже говорил: *сосредоточьтесь и лидируйте*. О том, как сосредоточиться, мы рассуждаем в этой главе. Что же касается лидерства — думайте

сами. В конце концов, роль лидера в вашей компании по праву принадлежит вам — так что играйте ее убедительно. Как говорил Шекспир, «весь мир театр, и люди в нем актеры».

Знание — это лишь исходное условие; по мере накопления опыта вы должны стремиться к принятию взвешенных решений, последовательно культивировать качества лидера.

КОШАЧЬИ РАЗБОРКИ — НЕ СПИ ЗА РУЛЕМ!

Грег слыл отъявленным неряхой. И дело не в том, как он ел, хотя и эта его черта зачастую становилась предметом обсуждения окружающих. Его небрежность в кодировании полностью соответствовала его наплевательскому отношению к своей внешности. По его понятиям, быстрое исправление кода сводилось к созданию очередной глобальной константы, передающей информацию между объектами, с попутным загаживанием своего рабочего места остатками тут же поедаемого хот-дога. Впрочем, мы, его коллеги, понимали в плане кодирования не больше его — на протяжении десятилетия доминирования DOS мы привыкли пользоваться любыми подручными средствами, лишь бы заставить программу работать. Объектно-ориентированная технология среди наших приоритетов в то время не значилась.

Мы только что приступили к работе над новой программой последовательной связи, и наш отдел продаж требовал поставить ее как можно быстрее. В те времена такого понятия, как отдел тестирования, еще не существовало — штат сотрудников проекта ограничивался кодировщиками, пытавшимися всеми силами успеть к срокам, установленным отделом продаж. Грег в нашей компании трудился уже довольно долго и, надо сказать, весьма успешно, поэтому к постоянному давлению привык. По крайней мере, мне так казалось. У него даже была репутация ценного сотрудника — в основном потому, что он единственный из нас всех умудрялся сопровождать код старых приложений, которые уже не продавались.

В отделе все стояли на ушах. Платформа Windows 3.0 только что появилась, и наши старания по части разработки приложений сочетались с попытками постичь идиосинкразическую природу интерфейса прикладного программирования Windows. И, между прочим, у нас это неплохо получалось. В нашем отделе у каждого программиста было собственное помещение, и в этих помещениях даже были двери, которые, как это ни странно, закрывались. Это и погубило Грега.

Однажды, показывая заказчикам условия, в которых мы работали, директор с вице-президентом появились в отделе разработки. Познакомившись с парой инженеров, группа направилась в кабинет Грега. Что же они увидели, открыв дверь? Грег мирно спал за столом, водрузив на него ноги. Повсюду были разбросаны пластиковые пакеты из-под еды, иногда с древними остатками самой еды, да и вообще комната представляла собой весьма унылое зрелище и вряд ли могла кому-то понравиться. Меня в тот момент там не было, поэтому я говорю с чужих слов.

А рассказали мне много интересного. Вице-президент по продажам настоял на том, чтобы уволить Грега немедленно. Директор согласился. Выбора у меня не было. Вскорости я сообщил Грегу неутешительные новости, после чего был вынужден наблюдать жалкую сцену мольбы о пощаде и последнем шансе. Я ему сказал, что это не в моей власти, что мне ужасно жаль, и еще до обеда его рабочее место очистили от мусора, а его самого выставили за дверь.

Какова мораль? Очевидно, она в том, чтобы не спать на работе. Но это еще не все. К сожалению, некоторые программисты не выдерживают нагрузки. Мы должны работать в высоком темпе, а с теми, кто за ним не поспевает, приходится прощаться. Это жестокая реальность. Надеюсь, у Грега все нормально, — ведь с тех пор я его ни разу не видел.

Кодовая полиция

Переместимся из XVI в XX век — как говаривал Элиотт, «Это один из вариантов конца света/Без треска, но с воем»[1]. Не стоит делать программы по этому принципу. Для того чтобы избежать этого, вам придется играть роль кодового полицейского. Если выражаться более знакомыми терминами, вам предстоит проводить регулярные критические обзоры кода, направленные на проверку соответствия архитектурной базе и принципам проектирования. Вспомним принцип, который я сформулировал в главе 2, — без регулярных проверок нельзя рассчитывать на хорошие результаты. Именно поэтому критические обзоры кода так важны.

В ходе проведения обзора вам предстоит столкнуться с двумя трудностями. С одной стороны, против вас работает время; с другой — программисты зачастую слишком ревностно относятся к попыткам редактирования своего кода. В конце концов, в сутках всего 24 часа. Пытаться изменить это обстоятельство бесполезно, поэтому единственный выход — учиться использовать время рационально. О том, как наиболее эффективно распоряжаться своим временем, говорится в главе 4, посвященной организованности.

Что касается недовольства вторжением в результаты своей деятельности, программистам придется смириться. Не стоит пугаться, что программисты отреагируют на такое вмешательство резко негативно. Это лишь признак сопричастности, а она, как известно, является необходимым условием создания качественного программного обеспечения. Хуже, если программист относится к вашей конструктивной критике индифферентно — из этого можно заключить, что конечный результат его не интересует. Толстокожесть техническому лидеру не повредит. Если вы слишком стеснительны, чтобы исправлять чужие ошибки, вы рискуете набить немало шишек — быть может, это вас образумит.

Следите за законностью

Вы вместе с вашими подчиненными должны работать так, как будто повязаны узами контракта. В качестве основных положений контракта при этом выступают коммерческие спецификации и ваше видение архитектуры; условия же контракта сводятся к принципам и методикам проектирования. Будучи программным полицейским, вы должны следить за корректностью разрабатываемого программного продукта, начиная от зародышевого состояния и вплоть до зрелости. Впрочем, довести программный продукт до состояния зрелости пока что не удается — при сегодняшнем технологическом уровне приложения останавливаются в своем развитии в подростковом состоянии. Ну, может быть, мы приближаемся к юношеству. Каждая компания по-своему уникальна, в каждой приняты собственные стандарты оценки зрелости продуктов[2]. Принципам оценки в программной инженерии посвящены многочисленные издания. Кейперс Джонс (Capers Jones) в своей книге «Applied Software Measurement»[3] утверждает, что наиболее успеш-

---

[1] T.S. Elliot, Collected Poems 1909-1962 (New York: Harcourt Brace Jovanovich, 1971), p. 82.
[2] См. Humphrey, op. cit., p. 5.
[3] Capers Jones, Applied Software Measurement (New York: McGraw-Hill, 1991), p. 1.

ные компании, работающие в сфере разработки программного обеспечения, отличаются шестью общими характеристиками.

1. Они проводят точные измерения продуктивности и качества программных продуктов.
2. Они тщательно планируют и оценивают программные продукты.
3. У них работают квалифицированные менеджеры и технические специалисты.
4. У них адекватные организационные структуры.
5. Они пользуются наиболее эффективными методами и инструментами разработки программного обеспечения.
6. Их сотрудники работают в достойных условиях.

Из всех этих характеристик наиболее важной Джонс считает первую. Именно здесь в полную силу проявляются преимущества критических обзоров кода.

Правила вам известны. Если бы вы их не знали, объяснить ваше продвижение по ступенькам административной лестницы было бы довольно трудно. Смысл критических обзоров кода состоит в донесении вашего опыта до менее бывалых подчиненных. Критические обзоры — это вам не контрольные работы в школе, за которые в дневник ставят оценки, а потом напротив них свои подписи ставят родители. Скорее их можно сравнить с университетскими семинарами, на которых проницательные студенты стремятся выявить наилучшие идеи и забраковать неудачные. Сотрудники, демонстрирующие неспособность учиться на критике, вряд ли смогут долго оставаться в нашей профессии. Быть может, такой подход покажется вам слишком жестким, но если не вывести теоретические основы разработки на практический уровень, будет страдать качество конечного продукта.

Наиболее распространенные нарушения

В своем кратком обзоре основных принципов проектирования я акцентирую ваше внимание на трех аспектах: стандартах, связности и взаимозависимости. Рассмотрим соответствующие нарушения.

Нарушение стандартов

Везде ли проставлены комментарии? Сможет ли с их помощью новичок, не знакомый со структурой данного модуля, в нем разобраться? Предположим, вы пытаетесь проследить последовательность исправления ошибки, — можно ли это сделать по комментариям? Комментарии должны разъяснять, почему код написан именно так и никак иначе, и как этого удалось достичь — причем основной упор следует делать на раскрытии вопроса «почему?». О том как разработчик достиг поставленной задачи, свидетельствует сам код; в то же время комментарии помогают проследить ход мыслей разработчика во время разработки модуля. Для того чтобы продолжить чье-то начинание, вы должны знать, почему ваш предшественник пошел именно тем путем, которым пошел. В этом вам помогут комментарии.

Соглашение по именованию. Следуют ли ему ваши подчиненные? Возможно ли, взглянув на имя переменной, с уверенностью судить о ее области действия? Отлаживать код, в котором для определения области действия предположительно неверного параметра приходится тратить по полчаса, невероятно трудно. Бывает и так, что имена процедур настолько длинны, что наводят на мысль о беспо-

лезности или даже вредности введения длинных имен файлов в Windows. Быть может, они, напротив, настолько коротки, что для расшифровки имен подпрограмм приходится прибегать к словарю? Между этими крайностями нужно найти баланс — кто знает, может быть, для этого вам придется провести урок грамматики и объяснить своим подчиненным, чем глагол отличается от существительного. Я понимаю, что префикс «get» в именах подпрограмм, извлекающих данные, утомляет; но его наличие безусловно свидетельствует о том, что они что-то извлекают. То же самое можно сказать о префиксе «set». Простота во многих случаях — синоним практичности.

Комментарии должны разъяснять, почему код написан именно так и никак иначе, и как этого удалось достичь — причем основной упор следует делать на раскрытии вопроса «почему?». О том как разработчик достиг поставленной задачи, свидетельствует сам код; в то же время комментарии помогают проследить ход мыслей разработчика во время разработки модуля.

Соглашение по именованию и комментарии к коду предоставляют нам возможность употреблять в процессе написания кода единый язык. Не позволяйте своим подчиненным прибегать к «технологии безмолвия». Заставьте их должным образом расставлять комментарии и присваивать имена. Далеко не всегда разработчики отказываются от комментариев из-за спешки — иногда они сами плохо понимают, что сделали (например, скопировав готовый код из библиотеки и понадеявшись на авось). Бывает, они копируют комментарии, сделанные автором библиотеки, к которой они обращаются. Это много о чем говорит. Конечно, в основном про комментарии забывают из-за неуверенности в результате или из-за элементарной лени. Несколькими пометками в коде ситуацию не исправить. Такие явления зачастую означают значительно более серьезную проблему, с которой столкнулся кодировщик и решать которую нужно на более высоком административном уровне.

Среди прочих распространенных нарушений стандартов — применение подпрограмм с несколькими точками выхода, а также введение ненавистного оператора `GoTo`. (Программистам VB эту привычку можно простить — благо в .NET операторы `Try`, `Catch` и `Finally` включены в библиотеку.) Еще один распространенный недочет — регулярное отсутствие обработки ошибок. Некоторые убежденные в своей непогрешимости кодировщики считают обработку ошибок пустой тратой времени. Действительно, в некоторых случаях перехватить ошибку вверху цепочки вызовов вполне достаточно; в то же время неумение выявлять потенциальные источники ошибок свидетельствует об отсутствии у кодировщика навыков стратегического мышления.

Возвращаясь к перечню отличительных черт различных пород программистов, приведенному в главе 1, замечу, что с его помощью удобно выявлять и другие нарушения, допускаемые вашими подчиненными. Этот список можно продолжать бесконечно в зависимости от конкретного кадрового обеспечения[1]. Я лишь хочу

[1] Специалистам по VB я рекомендую труд James D. Foxall, Practical Standards for Microsoft Visual Basic (Redmond, WA: Microsoft Press, 2000). Что касается других языков, выбор литературы обширен; взять хотя бы классическое издание по C — Steve Maguire, Writing Solid Code (Redmond, WA: Microsoft Press, 1993). Дополнительную литературу см. в библиографии.

заметить, что, исполняя роль кодового полицейского, вы обязаны фиксировать имена тех, кто нарушает правила.

Слабая связность и сильная взаимозависимость

Слабая связность и сильная взаимозависимость — две области нарушений, которые допустимо рассматривать совместно, поскольку наличие одного предполагает наличие другого. Имеет ли в процедурах место обширное ветвление? Сильную взаимозависимость между процедурами обычно называют ветвлением по входу-выходу. При высокой степени ветвления по выходу функционирование процедуры предполагает обращение к множеству других процедур, что, естественно, нежелательно. Ветвление по входу, напротив, оказывает положительное воздействие, так как свидетельствует о строгой инкапсуляции. Общую оценку серьезности нарушений по части связности и взаимозависимости можно дать исходя из степени несоответствия основным объектно-ориентированным принципам.

Есть ли возможность по результатам анализа кода составить диаграмму блоков, демонстрирующую иерархии объектов? Выглядят ли отношения на такой диаграмме логичными, или же стрелки, напротив, расставлены бессвязно? Можно ли определить родословную объектов? При проведении критического обзора кода без таких вопросов не обойтись. Ваша задача — выявить слабые места и добиться их усиления. Не пытайтесь, впрочем, править код самостоятельно — ведь когда сроки поджимают, появляется искушение сделать все самому. Не забывайте, что, доверив исправление проблем сотрудникам группы, вы добьетесь гораздо более серьезного результата. Этот процесс, конечно, может затянуться, но вы ведь понимаете: если нет времени решить задачу сразу и правильно, то времени на то, чтобы переделать результат, тем более не останется.

Скорый суд и неотвратимость наказания

Заметив нарушения, вы должны приостановить процесс кодирования, пока нарушители не исправят ситуацию. Чем дольше нарушения будут игнорироваться, тем выше вероятность того, что код придется отправить прямиком на помойку[1]. Когда разработчики обнаруживают халтуру своих коллег, они невольно опускаются до того же уровня — это человеческая природа, ничего не поделаешь. Если владение оружием предполагает употребление обоих рук, то грамотное проведение критических обзоров кода требует пресечения деятельности нарушителей за счет авторитета руководителя.

Писать и читать материалы о критических обзорах кода не составляет труда; на практике же все оказывается значительно сложнее. Всем нам известны факторы, влияющие на качество кода и превращающие процесс сопровождения в хроническую головную боль. Новички в деле руководства и лидерства нередко испытывают сложности, пытаясь перенести теоретические представления о своих действиях в практическую плоскость. Сопротивление происходит от неуверенности в своих лидерских качествах, и технические навыки, какими бы впечатляющими они ни были, в данном случае отходят на второй план. Одно дело обсуж-

---

[1] См. Hunt and Thomas, op. cit. — авторы обозначают хаотичность в разработке программных продуктов изысканной метафорой: «не надо жить с разбитыми окнами». Я более чем уверен, что эта книга обязательно должна занять достойное место в вашей библиотеке, — уж очень она хороша.

дать проблемы кодирования в компании приятелей, и совсем другое — решать их с позиции руководителя. Если все сотрудники осознают слабости кода, они ожидают проявления вашей инициативы. Попытайтесь предвидеть трудности и внушить себе необходимость адекватного реагирования на них. О том, что *нужно* делать при выявлении проблем, я рассказал предостаточно. Имейте в виду, что прочтения этих строк совершенно недостаточно для превращения *нужно* в *сделаю*. Чтобы это случилось, что-то должно измениться в вашем характере. Именно об этом я и намерен поговорить в оставшихся разделах главы.

Философия в действии

Философия без практики бесполезна. Видение и действие должны сочетаться гармонично, как рука и перчатка. Иначе говоря, действуйте по своим убеждениям. Многие люди, не исключая программистов и руководителей, знают, как нужно делать, но тем не менее не делают этого. Существует ли верный метод экстраполяции теоретических знаний на практику, способный предотвратить разрушительные последствия бездействия? Никакого секрета здесь нет — скорее, возможен некий диалог, способный убедить вас слезть с койки (то есть отказаться от праздности) и приступить к действию. Быть может, перечисление последствий бездействия, исходя из сформулированных в этой главе принципов, вам поможет.

Делайте то, что считаете нужным.

Некачественная или случайная архитектура приводит к таким последствиям:

- сверхтрудное сопровождение и низкая оперативность (а следовательно, финансовые потери) при введении новых свойств;
- нежелание программистов заниматься сопровождением кода вследствие чрезмерной усложненности системы и неуверенности относительно первоочередных действий при исправлении ошибок и введении новых свойств;
- размывание кода из-за лавинообразного роста числа объектов, за счет чего при увеличении размера исполняемых файлов сокращается производительность.

Ниже перечислены последствия неадекватного проектирования:

- из-за несоблюдения стандартов создается впечатление, что все объекты создавались разными программистами;
- модификация в одном месте приводит к нарушению в нескольких других;
- вследствие многократного дублирования кода для изменения любой функции требуется найти и также изменить все ее экземпляры.

Любой из этих списков можно было бы продолжить, но, держу пари, страху я на вас нагнал уже предостаточно. Надеюсь, вы серьезно напуганы и готовы сделать все, чтобы перечисленные неприятности вас миновали.

Может, это и так, но вот я, например, предпочел бы еще пораскинуть мозгами. В конце концов, в мире полно людей с благими намерениями, которые, тем не менее, либо действуют строго противоположным образом, либо вообще ничего не делают.

Конкретный пример философии в действии — Леонардо да Винчи

Я не большой поклонник школы позитивного мышления и все-таки несколько лет назад меня заинтересовала книга о Леонардо да Винчи. Ее автор вознамерился научить читателя мыслить как Леонардо. Поскольку Леонардо широко известен, позволю себе называть его просто по имени. Билл Гейтс истратил кучу денег (которые он предварительно выкачал из нас за свои программы) на оригиналы дневников Леонардо. Этим все сказано. Леонардо был невероятно умен. Если вы всерьез заинтересуетесь его биографией (а для этого мало прочитать единственную книжку из серии «помоги себе сам»), то быстро поймете, что его творческий гений направлялся жизненной философией. Эту самую философию Майкл Гелб (Michael Gelb) вкратце сформулировал следующим образом[1].

- Неизменно любознательный взгляд на жизнь и сильнейшее стремление к постоянному обучению.
- Привычка проверять знания через опыт, настойчивость и готовность учиться на ошибках.
- Последовательное совершенствование восприятия действительности органами чувств (в особенности, зрением) как средство обогащения жизненной практики.
- Готовность к восприятию неясностей, парадоксальных явлений и неопределенности.
- Уравновешивание науки и искусства, логики и воображения. Мышление «всем интеллектом».
- Воспитание изящества в движениях, «двуправорукости», физического здоровья и поведения.
- Признание и понимание взаимосвязи между всеми вещами и явлениями. Системное мышление.

Думаю, из Леонардо вышел бы добротный программный архитектор. То что он был замечательным художником и инженером, не вызывает сомнений. Принципы, которым он следовал, достойны всяческого уважения — они стимулируют творческое мышление и являют собой прекрасный пример для подражания.

Влияние Леонардо на современную ему технологию впечатляет. Трудно себе представить, как один человек смог выдумать все те изобретения, которые он набросал в своих дневниках, а частично и реализовал. Очевидно, в этом ему помогла его жизненная философия. Что делает ее такой действенной? Уверен, что ни одна из ее составляющих не сыграла такой роли, как сбалансированное сочетание различных векторов его мышления.

Взять, например, его тягу к знаниям в сочетании с терпимостью к неопределенности. Убежденный в том, что опыт — лучший учитель, он постоянно совершенствовал свои воззрения с тем, чтобы улучшить наблюдательность. Леонардо воспитал целую плеяду последователей, уделяя значительное время оценке их достижений. Быть может, он был первым, кто пользовался принципом системно-

---

[1] Хотите верьте, хотите нет, но я эту книгу прочел. Michael J. Gelb, How to Think Like Leonardo da Vinci (New York: Dell Publishing, 1998), p. 9.

го мышления — а ведь это именно то, к чему мы, специалисты по проектированию программных продуктов, неизменно стремимся. Этот принцип предполагает следование установленному плану вплоть до его окончательной реализации. Он имеет непосредственное отношение к тому, о чем я уже говорил, — к комплексной проверке архитектуры, предваряющей ее исполнение в виде подсистем-компонентов.

Из иных творений Леонардо мы обнаруживаем, что он был человеком, чрезвычайно крепким физически. Он не позволял телу тормозить работу ума. Быть может, это слишком личное, но подумайте: готовы ли ваши чресла к решению задачи, поставленной интеллектом? Если нет, придется вам поработать над физической формой, совершенствуя попутно мозговые кондиции. Вы и только вы способны понять, что в вашей рабочей деятельности препятствует следованию здоровой философии.

Крайне рекомендую на досуге почитать Леонардо — надеюсь, это поможет вам на практике проводить в жизнь стройную систему деятельности. Вполне возможно, побудить вас к действию смогут биографии современных лидеров нашей индустрии[1]. Полезно также ознакомиться с историями успеха деятелей других отраслей экономики. Удивительно, но факт: они в своей деятельности сталкиваются с теми же проблемами, что и мы. Короче говоря, биографическая литература — это неплохой выбор для чтения перед сном. По крайней мере, она поможет вам отдохнуть от повседневного чтения трудов по ADO 2.1, MTS MSMQ, VB, ASP, HTML 4.0 — видимо, их авторы считают, что заумная аббревиатура на корешке завлекает покупателя[2]. Кто знает, быть может, биография Джуди Гарланд (Judy Garland) напомнит вам, что как программист, руководящий другими программистами, вы не в Канзас-Сити.

Ложка дегтя

Умирая, Леонардо понимал, что труд его жизни остался незаконченным. Это жестокая реальность. Тем не менее я призываю вас стремиться к самосовершенствованию и, руководствуясь полюбившимися философскими воззрениями, каждую неделю делать немного больше полезных дел. Между знанием и действием в нашей рабочей действительности огромная пропасть. Это своеобразное состязание между эпистемологией и онтологией. Факт тот, что изначально мы предрасположены к бездействию; однако, в конце концов, происходит что-то, что заставляет нас воскликнуть: «Хватит с меня этих проблем в коде!» — и наконец сделать что-нибудь стоящее.

Перспективы

Давайте абстрагируемся от подробностей и оценим результаты вашей деятельности в роли технического лидера — довольны ли вы теми продуктами, которые появились на свет при вашем участии? Ниже я делаю краткий обзор вопросов, которые мы затрагивали в этой главе. Надеюсь, этот перечень поможет вам адекватно оценить степень своей полезности в области технического лидерства.

---

[1] Есть одна восхваляющая амбициозность книга, которая мне очень нравится. James Champy and Nitin Nohria, The Arc of Ambition (New York: Perseus Books, 2000).

[2] Кстати, это неплохой маркетинговый прием — по крайней мере, я такие книги покупаю.

- Удается ли вам последовательно играть роль технического лидера? Я совершенно не хочу сказать, что ваши идеи всегда должны быть на порядок лучше чужих, — просто ваши обязанности иные, к ним относятся отбор наиболее удачных идей и их практическая реализация.
- Предполагают ли варианты архитектуры ваших продуктов расширяемость и удобство сопровождения? Ответ на этот вопрос можно получить лишь тогда, когда первая версия продукта уже выпущена на рынок, а вы принялись за оценку ресурсов, требующихся для производства второй версии.
- Смогли ли вы за счет своих проектных решений обеспечить возможность многократного использования компонентов? Может ли любой сотрудник вашей группы взять модуль, разработанный другим сотрудником, и успешно справиться с его сопровождением или расширением функциональности? Ответы на эти вопросы свидетельствуют о ваших достижениях по части организации и руководства проектной деятельностью.
- Как вы обращаетесь с выявленными недостатками кода — устраняете их сразу на стадии разработки или ведете журнал фрагментов кода, требующих переработки?
- Способствует ли здоровая критика действий ваших подчиненных повышению их квалификации?
- Что вы предпочитаете — умные разговоры или практические действия? Помните ли вы, что нереализованные философствования бесполезны?

Руководствуясь принципами самоконтроля, вы должны регулярно задаваться этими вопросами. И не забывайте на них отвечать. Техническое лидерство взращивается на почве знания и питается готовностью учиться на собственных ошибках — в конечном итоге такая практика обязательно увенчается успехом.

Техническое лидерство взращивается на почве знания и питается готовностью учиться на собственных ошибках — в конечном итоге такая практика обязательно увенчается успехом.

Что дальше

Проводить философские воззрения в жизнь не так уж просто. В следующей главе, в которой мы поговорим об обратной стороне лидерства, у вас еще будет возможность поразмышлять о последствиях неверной философии лидера. Даже удивительно, сколько неприятностей способны доставить люди, руководствующиеся неподходящими философскими концепциями. Однако они, по крайней мере, уяснили, как превращать теорию в практику. Повторюсь — оценивать свои лидерские качества нужно по результатам каждодневной практической деятельности, ведомой внутренними убеждениями.

Глава 7

Закат лидера

Йода говорил: «Остерегайтесь темной стороны». Это предупреждение применимо и к нашей деятельности — у лидерства тоже есть обратная сторона. Иначе говоря, если отказаться от лексикона научной фантастики, лидерство бывает неэффективным и разрушительным. Судя по моему опыту, темными (то есть не излучающими свет, за которым могли бы следовать окружающие) личностями люди становятся благодаря стилю руководства, который просто не создает почву для развития лидерских качеств. Результатом неадекватных руководящих действий становится закат лидерства, который может повлиять как на вас, так и (что хуже) на вашего шефа. Подобный стиль руководства, оказывая воздействие на суждения, незаметно разъедает даже самых сильных лидеров. В результате эффективность лидерства сходит «на нет» из-за неадекватного стиля руководства.

Во всех предыдущих главах я исходил из вашей способности отказаться от действий, приводящих к неэффективному руководству, и проводить в жизнь методики, обеспечивающие истинное лидерство. Впрочем, некоторые люди неспособны к изменениям, и воспринимать их следует как поддержанный автомобиль — в состоянии «как есть», без гарантии и без возврата потраченных средств. Полагаю, что вы не относитесь к этой категории, — в противном случае, столкнувшись с тем, что я ставлю под сомнение обыденные представления о руководстве и лидерстве, вы просто забросили бы чтение. В то же время в определенной степени упрямство и нежелание меняться присутствует во всех людях — естественно, эти качества мешают должному исполнению нами своих обязанностей. По мнению Йоды, адептами «темной стороны» становятся те, кто пребывает во власти страха. Может, он и прав. Мне действительно кажется, что страх совершить ошибку — это одно из проявлений негатива, о котором пойдет речь в этой главе. Вы должны уяснить для себя, что страх ошибиться можно преодолеть и маскировать его непродуктивным стилем руководства совершенно не обязательно.

Обличие тьмы

Страх порождает самые что ни на есть темные явления. Примером тому — дьявольский период правления Гитлера. Благородная сама по себе немецкая концепция «лидера» (der Furer) превратилась в кошмар западной цивилизации. Эти зловещие страницы истории наталкивают меня на вывод: лидерство, если оно носит разрушительный характер, чревато страшными последствиями для окружающих. Аналогичным образом (хотя и в несколько более скромном масштабе), если группа разработчиков по вине своего лидера пойдет по неверному пути, следует ждать снижения производительности, ухудшения качества выпускаемых данной компанией программных продуктов и неспособности сотрудников достигать намеченных целей. Есть одна древнееврейская пословица (может быть, вы ее слышали), которая звучит примерно так: «недальновидность правителей губит людей». Вот что случается с теми, кто теряет нить руководства. Среди подчиненных начинается анархия, но ведь, как я говорил в главе 1, руководитель должен стремиться к тому, чтобы сотрудники его рабочей группы двигались в одном направлении. Именно поэтому я считаю лидерство основной темой этой книги. Когда лидер поддается дурным веяниям, он перестает быть таковым.

Получив представление об извращенных стратегиях лидерства, которые я описываю в этой главе, вы должны сопоставить их с собственной административной деятельностью и попытаться найти общие черты. За этим нужно следить — перенимая негативные элементы таких стилей, вы снижаете свой профессиональный уровень. Как известно, сформулированная проблема наполовину решена. Вторая половина требует больших усилий, но, надеюсь, осознание пагубности неверной стратегии руководства и ее отрицательного влияния на подчиненных вас мобилизует. В таком случае вы сможете с решимостью перейти к действиям по решению проблемы, то есть скорректировать свои действия в роли руководителя. Полагаю, на это у вас хватит сил — в противном случае вы вряд ли добрались бы до этих строк. Прислушайтесь ко мне — в конце концов, мне самому каждый день приходится бороться с обратной стороной лидерства.

Негативные эталоны в менеджменте

Полагаю, знакомясь с материалом главы 1, вы изрядно позабавились по поводу описания различных типов программистов. В этой главе я обращаюсь к аналогичной схеме повествования, хотя на этот раз она вряд ли доставит вам соизмеримое удовольствие. Итак, вам предстоит ознакомиться с негативными эталонами в психологии — благо о том, что они собой представляют в области проектирования, у вас уже есть некоторое представление. Назначение негативных эталонов — показать, как не стоит себя вести. Если среди перечисленных черт вы обнаружите сходство с собственной моделью поведения, попытайтесь ее скорректировать. Начальников следует остерегаться и быть осторожным в оценках. Быть может, это слегка упрощенный совет, но пока что он вполне достаточен. В контексте проблем, которые обсуждаются в этой главе, мы еще успеем поговорить о преодолении трудностей и корректирующих стратегиях.

Многие руководители, которым, по идее, следует стремиться к приобретению лидерских качеств, на деле демонстрируют в своем поведении смесь стилей. Об

этих стилях, которые проявляются с той или иной силой, и пойдет речь. Анализируя порочные стили руководства, я не утверждаю, что они олицетворяют абсолютное зло, я вообще стараюсь воздерживаться от моральных оценок. Значительно важнее другое: по тем или иным причинам отдельные руководители перенимают отрицательные качества деструктивных стилей руководства. Они со всей очевидностью проявляются в рабочей среде и снижают качество лидера. Знание перечисленных ниже негативных эталонов не означает, что вы должны наставлять на путь истинный всех, кто им следует, — ваша цель в том, чтобы выявлять соответствующие типы людей и уметь с ними общаться. С другой стороны, вы должны подавлять эти качества в самом себе. Вся эта книга преследует одну цель — направить вас на путь лидерства. Поймите, других людей вам не исправить. Это очевидное, на первый взгляд, обстоятельство очень помогает в процессе взаимодействия с окружающими.

Поймите, других людей вам не исправить. Это очевидное, на первый взгляд, обстоятельство очень помогает в процессе взаимодействия с окружающими.

Заимствовав некоторые идеи у апологетов применения негативных эталонов[1], я решил при описании порочных стилей руководства придерживаться следующей структуры.

- **Симптомы и последствия.** С точки зрения феноменологии, понять истинные намерения людей, следующих в своем поведении негативным эталонам, невозможно — зато их поведение вполне идентифицируемо. Если вы сами поддались влиянию одного из негативных эталонов, не забывайте о своем преимуществе, которое выражается в том, что у вас есть все средства контролировать собственную сознательную деятельность.
- **Вариации.** Имеется в виду, что одна и та же проблема может выражаться поразному. Тем не менее она неизбежно ведет к ухудшению лидерских качеств.
- **Исключения.** Обо всех исключениях я буду говорить отдельно. Существуют ли ситуации, в которых применение стиля руководства, не способствующего развитию лидерства, допустимо? Да, существуют, но об этом позже.
- **Решения.** Здесь я объясню, как скорректировать неверный стиль. Эти рекомендации, естественно, применимы только к вам самому, поскольку, как я уже говорил, заставить измениться окружающих невозможно. В то же время понимание способов корректировки стиля руководства поможет вам как можно быстрее вернуть утраченные лидерские позиции. Решения наиболее общего характера рассматриваются в заключительной части главы; впрочем, по мере изложения материала я иногда буду акцентировать ваше внимание на методах корректировки, применимых в контексте конкретного стиля.

Описание каждого негативного эталона начинается с заголовка, в котором я пытаюсь в сжатой форме изложить его суть. Все негативные эталоны я развел по отдельным разделам — они настолько серьезны, что их надо изучать по отдельности.

[1] См. Brown et al, AntiPatterns in Project Management, op. cit.

Мелочная опека

Отдельные признаки мелочной опеки — в высшей степени порочного стиля руководства — демонстрирует великое множество руководителей. Упадочным этот стиль считается по одной простой причине — он являет собой полную противоположность качественному руководству, которое основывается на делегировании и проверке. Деятели, увлекающиеся мелочной опекой, сводят с ума не только окружающих, но и самих себя. Признаться, все мы в определенных условиях этим грешим. Ну вот, к примеру, классическая ситуация — цикл разработки подходит к концу, как вдруг выясняется, что один из ваших сотрудников забыл выполнить данное ему задание. Если времени в обрез, а вокруг никого, кому это задание можно было бы перепоручить, вы, скорее всего, не удержитесь и возьметесь за него сами. Скажете, это простительное исключение? Может быть, но имейте в виду — действуя подобным образом, вы приучаете своих подчиненных к тому, что за невыполненную работу никаких взысканий не последует — вы просто сами все доделаете. Хорошо ли это? Думайте сами.

Как правило, мы, технические лидеры, приучены к активной и напряженной работе, в связи с чем нам иногда кажется, что с поставленными задачами мы справимся лучше, чем те, кому мы их делегировали. В конце концов, очень часто руководителями становятся именно те сотрудники компании, которые проявили себя высокопрофессиональными инженерами. Не забывайте, впрочем, что кодирование не относится к числу наших первоочередных обязанностей. Руководитель призван обеспечить решение задачи силами своих подчиненных. Первое, что должен сделать лидер, — это создать условия для полноценного делегирования. Тот, кто увлекается мелочной опекой, дискредитирует себя как лидера и как руководителя. Я даже не допускаю мысли о том, что руководитель, чья деятельность подчинена стилю мелочной опеки, может быть лидером — на это у него просто нет времени. Все, что он делает, — это вмешивается в действия своих подчиненных. Если вам так нравится, называйте их «мелочными лидерами». Свидетельства их лидерских качеств можно разглядеть разве что под микроскопом.

Руководитель призван обеспечить решение задачи силами своих подчиненных. Первое, что должен сделать лидер, — это создать условия для полноценного делегирования.

Представители этого типа руководителей встречаются в нашей отрасли чаще всего. Их можно разделить на несколько типов.

- **Всезнайка.** Такой деятель искренне верит в то, что ему известно все о его работе, о работе компании, о конкретных заданиях программистов. Он считает себя эрудитом — несмотря даже на то, что этот термин, как мне кажется, исчерпал себя в веке эдак семнадцатом. Я бы не отказался быть эрудитом, да и вы, полагаю, тоже, но в сегодняшних условиях это просто невозможно — объем накопленных со временем технических знаний слишком велик. Некоторые считают, что навыки веб-серфинга делают их умнее и информированнее. Никто не оспаривает роль Интернета как одного из инструментов познания, но ведь на дворе нынче век информации, но никак не век знаний. Ни серьезная

технологическая подготовка, ни невероятная сноровка в разгадывании кроссвордов не дают вам права претендовать на единоличное выполнение всего комплекса заданий.

- **Диктатор.** Девиз этих деятелей — «я прав, потому что я прав». Диктатор пытается навязывать своим подчиненным конкретные методы решения поставленных перед ними задач. На самом деле всем нам не мешает тщательно следить за тем, чтобы не пойти по этому пути. Лидерство действительно предполагает демонстрацию сотрудникам возможных способов работы, но в этом следует соблюдать осторожность, поскольку лишать подчиненных чувства сопричастности к созданию продукта недопустимо. Диктатор создает вокруг себя атмосферу животного ужаса, которая не позволяет сотрудникам в полной мере проявлять свои способности, — все с трепетом ожидают очередного указания сверху и пытаются угадать последствия малейших проявлений независимости. Даже в политике случаев, когда диктаторство можно было бы признать уместным, ничтожно мало — что уж говорить о бизнесе! Настоящий лидер должен создавать условия, в которых разработчики могли бы гордиться результатами своих трудов, а для того чтобы их гордость воплощалась в хороших результатах, им следует предоставлять определенную свободу выбора средств достижения цели. Тем не менее хороший руководитель устанавливает некоторые пределы свободы. Диктатор же ее просто-напросто подавляет.
- **Генерал.** Руководитель этого типа хорошо усвоил уроки школы «командования и управления» в разработке программных средств. Генералы обнаруживают значительное сходство с диктаторами, проявляя в своей деятельности еще большую жесткость. Было время, когда военная модель в бизнесе была распространена довольно широко. В военных формированиях, где все носят униформу, которая подтверждает принадлежность людей к огромному механизму, управляемому командами, эта схема имеет полное право на существование. Кстати сказать, такой стиль руководства характерен для многих крупных программных проектов, разрабатываемых в гигантских корпорациях или в правительственных структурах. Именно в этой среде зародилась методика программной инженерии. По-моему мнению, относительно сегодняшних программистов этот метод себя не оправдывает — между выпасом котов и командованием войсками наблюдается существенная разница. Программисты не носят униформу (если, конечно, не считать таковой их дурацкие футболки) и обожают бравировать своей независимостью. Если вы вообразили себя генералом, а своих подчиненных считаете солдатами, имейте в виду: битвы с противником вам придется вести одному. Солдаты, скорее всего, будут отсиживаться где-нибудь в углу, спокойно попивая кофе. Противостоит такому подходу стиль сотрудничества — и, вы знаете, он имеет широкое употребление, просто потому, что он правильный. Во время войны сотрудничество редко имеет место. В области разработки программного обеспечения, напротив, трудно себе представить ситуацию, в которой установлению отношений на основе сотрудничества что-либо воспрепятствовало бы.

Первопричина мелочной опеки кроется в убежденности руководителя в том, что никто не может выполнить работу лучше него. Прибавьте к этому страх допустить ошибку, и вы поймете, почему так много руководителей попадаются в эту

ловушку. Я не сомневаюсь, что вы хотите преуспеть в своей деятельности, но подумайте, какую цену вы готовы за это заплатить. Какой смысл было нанимать сотрудников, если вы намерены взять на себя все их обязанности? Если практиковать мелочную опеку достаточно долго, вы рано или поздно достигнете предела продуктивности. Этот предел определяется, с одной стороны, вашей способностью делать работу за других, с другой — терпением ваших сотрудников, осознающих невостребованность своих талантов.

В табл. 7.1 сравниваются характеристики руководителя, практикующего мелочную опеку, и полноценного лидера.

Таблица 7.1. Мелочная опека и лидерство

Руководитель, практикующий мелочную опеку	Лидер
Что я должен сегодня сделать?	Как помочь подчиненным с достоинством справиться с их задачами?
Что, если у меня ничего не получится?	Как совместными усилиями мы можем свести к минимуму вероятность неудачи?
Почему сотрудники не подчиняются моим указаниям?	Как бы почетче формулировать свои наставления?

Вероятно, анализируя собственный стиль руководства, вы сможете выявить и другие черты, отличающие лидерство от мелочной опеки. Все мы время от времени впадаем в эту крайность. Обратите внимание на то, где в табл. 7.1 встречаются местоимения «я», «меня», «моим». Увлекаясь мелочной опекой, мы неизбежно замыкаемся на себе; полноценный лидер, напротив, стремится к конструктивному сотрудничеству с окружающими. Освободиться от оков мелочной опеки помогает осознание ценности ваших сотрудников и пользы от взаимодействия с ними. Не слишком затягивайте узду. Не забывайте, что у вас есть персонал — дайте ему спокойно работать.

КОШАЧЬИ РАЗБОРКИ — САМОДУР

Эта история произошла в одной небольшой программно-консалтинговой компании лет десять назад. Работала фирма стабильно, принося владельцу солидный доход. Ее немногочисленные, но преданные сотрудники были довольны объемом работы, заключавшейся в решении проблем всего нескольких заказчиков. Но хозяину этого показалось мало. В конце концов, это вполне объяснимое желание — увеличить масштаб деятельности компании и, соответственно, расширить клиентскую базу. «Либо вырастем, либо — гори все синим пламенем» — вот принцип, которым он руководствовался. Итак, он начал привлекать к деятельности компании консультантов, с тем чтобы они помогли расширить бизнес. Поскольку к своим сотрудникам он предъявлял довольно высокие требования, подобрать нужного человека было непросто. Но наконец подходящая кандидатура нашлась — новый консультант согласился взять на себя обязанности оперативного управления, подчиняясь при этом воле владельца. В теории план казался безусловно выигрышным.

Новоиспеченный менеджер с удовольствием взялся за исполнение новых обязанностей — он приступил к комплексному изучению деятельности компании, познакомился с сотрудниками и проектами, которые они вели. Хозяин, строя радужные планы по расширению бизнеса, принялся, в свою очередь, искать новых клиентов. Несколько месяцев дела выглядели как нельзя лучше и позволяли надеяться на блестящий результат.

Через некоторое время новый менеджер по развитию ввел в рабочее расписание сотрудников еженедельные совещания. Кроме того, он разработал для внутреннего пользования программу, позволявшую отслеживать текущие задания программистов и конечные сроки проектов. Сотрудники отнеслись к этим инициативам с пониманием и стремились придерживаться новых процедур. Раньше хозяин руководил процессом с помощью распоряжений, формировавшихся по большей части интуитивно. Теперь же, по мере роста, появилась необходимость в введении более формализованного подхода. Естественно, хозяин присутствовал на всех еженедельных совещаниях. И вот здесь начались трудности.

Многие решения и процессы, проводившиеся новым менеджером, владелец публично подвергал сомнению на совещаниях, вследствие чего они нередко задним числом «задвигались». Через несколько месяцев, когда совещания доказали полную нежизнеспособность, менеджер окончательно понял, что он здесь лишний. Сроки сдачи проектов по-прежнему не соблюдались, а программисты не могли понять, кто же, наконец, ими руководит.

В период реформирования штат пополнился несколькими новыми сотрудниками. При этом процесс подбора персонала превратился в сущий кошмар. Стоило менеджеру найти достойного претендента, как владелец его с завидной настойчивостью забраковывал. Из-за самодурства хозяина немногочисленные новички вскоре покинули компанию. Единственной областью, в которой хозяин предоставил менеджеру полную самостоятельность, было увольнение. И на том спасибо — по крайней мере, в этих вопросах менеджер мог быть уверен в том, что его решения не будут ставиться под сомнение.

Вскоре менеджера увлекли новые перспективы. После непродолжительных раздумий он решил сменить работу — благо на новом месте ему была предоставлена возможность полностью раскрыть свои способности.

Эта история — классический пример мелочной опеки, которой в данном случае увлекался владелец компании. История демонстрирует губительное воздействие на подчиненных политики руководителя, опасающегося «ослабить вожжи». Ощущение невостребованности и сознание невозможности в полной мере реализовывать свои навыки и высказывать мнения побуждает людей менять работу.

Советы тем, кто увлекается мелочной опекой

Существуют ли обстоятельства, в которых мелочная опека допустима? Вероятно, да — в ночь перед сдачей проекта при условии, что вы в силах быстро внести исправления. Во всех прочих случаях этот стиль ограничивает деятельность лидера. Лидерство ориентировано на привитие подчиненным собственной мотивации. В частности, лидер должен культивировать в сотрудниках чувство сопричастности (как успехам, так и неудачам) и предоставлять им возможность творческих начинаний. Мелочная опека исключает такие перспективы.

Предположим, вы поймали себя на следовании этой схеме. Как исправиться? Во-первых, попробуйте понять, откуда берется побуждение к мелочной опеке. Быть может, ваши сотрудники действительно так бездарны, что не справляются со своими обязанностями? Если это так, наберите новый штат. Если же к тотальному контролю над действиями подчиненных вас побуждает страх не достичь цели, придется вам провести серьезный анализ своих ожиданий. Никто не любит ошибаться, но, тем не менее, иногда это случается со всеми. Это реальность — восприятие ее в таком качестве помогает сбалансировать стиль рабочей деятельности.

Как скорректировать собственные ожидания, чтобы вероятность неудачи не казалась такой обескураживающей? Полагаю, для этого вы должны приучить себя анализировать действия, направленные на достижение успеха, не менее тщательно, чем собственные результаты. Люди всегда ошибаются; хороший лидер же ошибается только тогда, когда приложенные им усилия оказываются недостаточными. Неудачный результат простителен, а вот недостаточные усилия — это смертный грех. Приложите все усилия к тому, чтобы настроить сотрудников на продуктивную деятельность, — не распространяйте на них свои опасения о возможности неудачного исхода. Акцентируйте их внимание на сладости успеха, а не на горечи поражения. На самом деле, напоминать людям о незавидных последствиях неудач не стоит. Они это и так понимают. Сконцентрируйтесь на позитивной цели, и пусть ваши подчиненные спокойно работают в этом направлении.

Неудачный результат простителен, а вот недостаточные усилия — это смертный грех.

Доверие, на первый взгляд, плохо укладывается в контекст ограниченного по времени процесса разработки программных средств — деятельности, в которой зачастую даже малейшая оплошность способна привести к катастрофе. Ответ на вопрос: «Доверяете ли вы своим сотрудникам?» — неоднозначен. У вас в отделе могут работать самые компетентные программисты, однако отношения доверия можно построить только по прошествии длительного времени. Доверие предполагает риски: риск неудачи, риск разочарования и прочие не слишком приятные эмоции. Построить доверительные отношения, разумеется, непросто, — но, с другой стороны, они устраняют основную предпосылку мелочной опеки. На это уходит время, которого у вас не так много. В то же время отказ от мелочной опеки увеличит ваши временные резервы.

Построение доверительных отношений между руководителем и подчиненными проходит в двух направлениях. Проявляя надежность, поддержку и готовность решать их проблемы, вы заставляете подчиненных доверять вам. Выстраивание вашего доверия к ним проходит несколько этапов. Сформулируйте план-минимум — дайте каждому небольшое, не слишком важное задание и предоставьте возможность справиться с ним самостоятельно. Чем большую независимость в работе они будут проявлять, чем меньше им будет требоваться ваше непосредственное наблюдение, тем сильнее вы будете им доверять и тем более востребованными они себя почувствуют. Повторюсь — на выстраивание доверительных отношений уходит время, но, поверьте мне, результаты того стоят.

В компании Radio Shack практикуется замечательный принцип обучения персонала. Его можно сформулировать так: «делайте то, что нужно делать, даже если никто не смотрит». Именно к этому вы должны стремиться. Ваши подчиненные должны работать как следует просто потому, что они ответственны и мотивированы. По мере развития доверительных отношений между вами и вашей командой вы приучите сотрудников к мысли, что правильные действия приводят к наилучшим результатам — как для них самих, так и для компании в целом. Лидер должен постоянно стремиться к созданию такого рода условий; те же руководите-

ли, которые увлекаются мелочной опекой, напротив, препятствуют их формированию.

Что делать, если признаки мелочной опеки демонстрирует ваш собственный начальник? У вас два варианта действий: либо позволить ему взвалить на собственные плечи весь груз работ, либо доказать, что вы и сами можете справиться с задачами. Вполне может быть, что у вас так и не появится шанса проявить себя, но такая ситуация встречается довольно редко. Если уж вас наняли для исполнения конкретных функций, то, по крайней мере, одну возможность показать свою дееспособность вы получите. Если ваш начальник, отличающийся склонностью к мелочной опеке, ставит перед вами какую-то задачу, приложите все усилия к тому, чтобы решить ее лучше, чем он ожидает. Большинство руководителей, практикующих мелочную опеку, все-таки обращают внимание на выдающиеся усилия и соответствующие результаты. Они порабощены страхом неудачи, но если вам удастся приучить их к успеху, опасения, скорее всего, развеются. Если вы сможете наладить с начальником доверительный диалог, дайте ему знать, что вы постараетесь приложить к решению максимум усилий и ему совсем не обязательно постоянно стоять у вас за плечом. Доказывая начальнику свою компетентность и способность работать на его благо, вы, вероятно, сможете освободиться от оков его мелочной опеки.

Неорганизованные руководители

Неорганизованный руководитель обнаруживает недостаток организационных навыков, которые иногда, к тому же, сочетаются с нехваткой практического мышления.

Таких менеджеров иной раз называют выразительно пустоголовыми, раздолбаями, без царя в голове и т. д. Неорганизованные руководители в рабочее время думают о чем угодно, но не о работе. Иногда, впрочем, неспособность сосредоточиться объясняется недостатком опыта. Такие деятели часто кажутся забавными, но стоит только возложить на них ответственность за что-либо, как они сразу стараются от нее избавиться — а ведь это явление в процессе разработки программных средств представляет собой серьезную опасность.

Мне доводилось наблюдать за действиями персонажей, которых можно отнести к нескольким подтипам рассматриваемого стиля. В реальности их, конечно, больше, но я описываю только те, которые приходят на ум.

- **Скарлетт О'Хара.** Главная героиня классической киноленты 1939 года «Унесенные ветром», сталкиваясь с проблемой, требующей незамедлительного решения, любила говорить: «Я подумаю об этом завтра». Сразу уточню: совершенно не считаю, что рассматриваемый негативный эталон представлен исключительно женщинами. На самом деле мне доводилось работать только с руководителями-мужчинами, обнаруживавшими черты этого стиля, и, честно говоря, я не знаю в нашей области ни одной женщины-менеджера, которая грешила бы тем же. Деятели, работающие по данной схеме, либо не могут, либо не хотят концентрироваться на текущих задачах и решениях. Иногда им кажется, что задачи слишком сложны; в иных случаях они не принимают никаких решений по той лишь причине, что боятся совершить ошибку. Как вы уже,

несомненно, знаете, отказ от принятия решения зачастую равноценен санкции на его принятие другими людьми; иногда, что еще хуже, дальнейший ход действий обусловливается последствиями непринятых решений.

- **Временщик.** Имеется в виду человек, нацеленный в первую очередь на продвижение вверх по карьерной лестнице. Такие деятели, думающие исключительно о высоком, считают занятия менеджментом среднего звена не более чем времяпрепровождением. Большую часть своего времени они предпочитают тратить не на работу над проектом, а на демонстрацию окружающим своих амбиций. Сталкиваясь с необходимостью принятия решения, они обычно начинают активно интересоваться чужим мнением. Спору нет, лидер может интересоваться позицией окружающих, но в то же время у него должно быть собственное четкое мнение — прежде всего потому, что он совмещает анализ текущих задач с долгосрочным планированием. Временщики крайне редко уделяют достаточное внимание сотрудникам и проектам. В этом состоит основное противоречие, характерное для данного типажа. С одной стороны, эти люди хотят преуспеть и обеспечить тем самым профессиональный рост; с другой же — они обнаруживают неспособность к концентрации внимания, что не позволяет им достигать первой цели. Одними лишь мечтами о блестящем будущем его наступления не приблизишь. Лишь понимание того, что каждодневная работа есть дело вашей жизни, помогает сделать прожекты реальностью.
- **Новичок.** Это совсем неопытный руководитель, который вовсе не знает, с чего начать. Такие деятели и не помышляют о лидерстве; свою цель они видят в том, чтобы обуздать все административные тонкости своей деятельности. Некоторые из них даже считают, что лидерство определяется не достижениями, а одной лишь должностью. Руководители такого типа испытывают серьезные проблемы по части расстановки приоритетов, поскольку все задачи кажутся им в равной степени сложными. Иногда они приходят к тому же результату, что и двигатели самолета, пытающегося набрать высоту без достаточного ускорения, — то есть просто глохнут.

Последствия деятельности неорганизованных руководителей весьма разнообразны — в частности, они срывают сроки сдачи проектов и вносят в души подчиненных полный разброд относительно конкретных задач. Многие неорганизованные руководители имеют обыкновение в огромных количествах генерировать новые идеи, в связи с чем выполненный наполовину проект быстро оказывается устаревшим. Иногда неорганизованные руководители относятся к своим обязанностям совершенно несерьезно. Мне доводилось сталкиваться с руководителями, которые постоянно что-то увлеченно обсуждают (как правило, это «что-то» не имеет никакого отношения к их повседневной деятельности) — и, несмотря на их очевидное обаяние, после беседы задаешься вопросом: «К чему это все?»

Никто не спорит — хохмить можно и в рабочее время, однако если это подменяет деятельность по организации работы сотрудников, в конце концов будет не до шуток.

В выдающейся работе под названием «AntiPatterns in Project Management» высказывается следующая мысль:

> «...многие руководители проектов, не имеющие достаточного опыта лидерства, полагают, что руководство — это деятельность чисто формального характера.

Причины внезапной деградации проекта крайне разнообразны. Это может быть неадекватное управление критическим событием, неопытность или неумелость руководителя проекта, а также проведение в жизнь высших руководящих решений человеком, который, во-первых, не имеет достаточного представления о проекте, во-вторых, не уполномочен принимать сколько-нибудь значимых решений. Планы и процессы разработки программных средств не могут предусмотреть все возможные варианты развития событий; соответственно, в нестандартных ситуациях от них допустимо отступать. Руководитель проекта, равно как и другие руководители, должен демонстрировать изрядную гибкость — с тем, чтобы в случае выявления рисков справляться с ними практично и эффективно. Хаос, когда он уже начался, очень трудно остановить»[1].

Принимая необдуманные решения (а взвешенность решений характерна только для опытных лидеров), неорганизованные руководители способствуют образованию тупиковых ситуаций, допуская промедление, следствием которого оказывается хаос.

Неорганизованные менеджеры часто допускают промедление, следствием которого оказывается хаос.

Что, вы тоже неорганизованны? Отчего? Оттого ли, что вы боитесь принимать решения, или же вы просто путаетесь в тонкостях своей работы? Обе проблемы весьма распространены, а решаются они за счет повышенного внимания к организационным аспектам своей административной деятельности. Рекомендую вам в этом контексте еще раз обратиться к материалу предшествующих глав. В них я разъясняю основы выстраивания стратегии лидерства. Со временем, если вы будете упорно совершенствовать свои навыки, вы обязательно преуспеете.

Если вы намерены продолжить свой путь к вершинам менеджмента, имейте в виду: для этого успех должен войти в привычку, что невозможно без должного внимания к текущим задачам. Ни умозрительные схемы, ни прожекты не приведут вас к повышению. Этого результата можно достичь только за счет успешной деятельности; при этом чем дальше вы будете совершенствоваться в исполнении своих рабочих функций, тем более взвешенными станут ваши амбиции. Крайне трудно найти компанию, в которой повышения происходят только лишь по желанию претендента.

Никаких оправданий неорганизованному стилю руководства мне в голову не приходит. Причин, по которым формируется такая модель поведения, предостаточно, и, поверьте мне, время от времени любые руководители теряют концентрацию, однако настоящие лидеры умеют ее восстанавливать. Впрочем, восстановиться не так уж просто; иногда для этого требуется, скажем так, «перезарядить аккумуляторы». Разумно в такой ситуации взять отпуск или выделить выходной день на «восстановление умственных способностей». Для того чтобы вернуть способность контролировать тонкости процесса, любые средства хороши.

Что делать, если неорганизованного стиля придерживается ваш собственный руководитель? Попробуйте обратиться к технике «мелочной опеки наоборот».

---

[1] Brown et al, AntiPatterns in Project Management, op. cit., p. 39, 40.

Предлагайте разумные решения задач. Делайте то, что считаете нужным, — в особенности если задачи распределены недостаточно четко. Если ваш руководитель дистанцируется от проблемы, у вас будет больше времени. Пользуйтесь этим временем с умом. Кто знает — может быть, вас повысят. Впрочем, это не есть ваша основная задача; главное — должным образом справиться с решением проблем, стоящих перед вашим отделом. Если начальник не хочет работать, инициатива переходит к его подчиненным — при условии, конечно, что им не все равно, как идут дела в компании.

Гений не на месте

Довольно часто в нашей индустрии в менеджеры производят блестящих программистов, неспособных к руководящей деятельности и уж, тем более, неспособных к лидерству. Такой программист может весь день сидеть за клавиатурой и «строгать» прекрасный код, но в том, что касается передачи знаний окружающим, он полный ноль. Проводя проектное совещание, гений демонстрирует чудеса на электронном табло, но объяснить слушателям суть понятий, которые он на нем с такой легкостью иллюстрирует, у него не получается. Консенсус он понимает как единодушное согласие со своими предложениями.

Гении бывают самые разные. Я не намерен воспроизводить характеристики всех подтипов (слишком их много); скажу лишь, что всем гениям свойственно одно и то же качество — исключительная способность к концентрации. Основной акцент они обычно делают на технологии, отодвигая кадры на второй план. Именно это обстоятельство мешает им стать полноценными лидерами. Спору нет, руководитель обязан обращать внимание на вопросы технологии, но основным приоритетом в его деятельности должны быть люди, которые работают с технологией. Код не возникает сам по себе. Его создание есть результат умственной работы программистов, стимулируемых интеллектуальной тиранией руководителей.

Гении в ранге руководителей имеют обыкновение разрабатывать для применения подчиненными целые программные шаблоны, которые, по их мнению, необходимо задействовать во всех без исключения продуктах. Зачастую такие шаблоны оказываются слишком сложными для применения в небольших проектах, однако гении, несмотря ни на что, настаивают на их применении, объясняя это необходимостью «поддерживать единую кодовую базу». Гении создают собственные шаблоны даже в тех случаях, когда продукты, способные выполнять те же функции и снабженные толковой документацией, присутствуют на рынке в большом количестве. За многие годы моей профессиональной деятельности мне не раз приходилось сталкиваться с гениями, и, надо заметить, я научился у них самым разным техническим приемам. Но вот по поводу менеджмента ничего путного они мне сообщить не смогли, за исключением, пожалуй, фактических сведений для введения данного негативного эталона, выражающего неверное приложение исключительных умственных способностей.

Один из гениев, с которым я столкнулся на своей предыдущей должности, умудрился разработать такой шаблон, в котором для выгрузки пользовательского интерфейса требовалась цепочка вызовов из пятнадцати процедур. Шаблон этот создавался в расчете на один из предшествующих продуктов, но мой гений, тем не

менее, настаивал на том, чтобы использовать его и в весьма скромном проекте, хотя последний, откровенно говоря, совершенно не соответствовал выбранному решению. На изучение принципов функционирования шаблона и способов взаимодействия с ним у меня ушло больше времени, чем на написание нового кода и доведение продукта до стадии альфа-тестирования. Надо сказать, остальные члены команды испытывали те же проблемы. Мы все были поражены блестящим интеллектом нашего руководителя и не могли устоять под прессом его методологических предложений. При этом большинство наших продуктов так и не добралось до стадии бета-тестирования, поскольку на их разработку уходило слишком много времени и к моменту завершающего этапа коммерческая потребность в продуктах исчезала. Он был действительно очень умен, но как организатор человеческих ресурсов и процессов никуда не годился.

Такого рода менеджеры, как правило, разрушают те административные процессы, с помощью которых их подчиненные ранее достигали успехов. Гении рассматривают администрирование как работу «для других», недостойную их внимания. Руководители высшего уровня, как правило, считают гениев прекрасными лидерами, объясняя свою точку зрения их выдающимися способностями. Чем больше наблюдатель удален от повседневных процессов, тем выше он склонен оценивать достижения гениев, не замечая их откровенных недоработок, происходящих от неэффективного распоряжения человеческими ресурсами и пренебрежения к административным процедурам.

Гении рассматривают администрирование как работу «для других», недостойную их внимания.

Что, усматриваете в этом описании свои черты? Если так, поймите, наконец, что ваши знания не находят себе нужного применения. Постарайтесь, не отказываясь от акцента на технологии, обращать больше внимания на кадры, без которых никак не обойтись. Для сравнения рассмотрим табл. 7.2.

Таблица 7.2. Гений и лидер

Как мыслит гений	Как мыслит лидер
Какая методика разработки программного обеспечения может помочь решить данную проблему?	Как мне привлечь других сотрудников к формулированию методик решения проблем?
Сколько еще языковых средств мне удастся изучить в этом месяце?	Какую образовательную подготовку стоит пройти моим сотрудникам, чтобы они смогли повысить показатели продуктивности?
Сложность является препятствием только для невежд	Сложность оправдывается лишь в том случае, если все участники проекта в достаточной мере разбираются в вопросе и внесли свою лепту в создание системы

В ваших силах расширить ваши горизонты и поставить вопросы кадрового обеспечения на центральное место в процессе решения бизнес-задач. Ваши подчиненные в любом случае обращаются к технологиям, и здесь вы сможете им

серьезно помочь; не стоит в то же время забывать, что, пренебрегая человеческим фактором, вы теряете лидерство.

Что если вам приходится работать под началом типичного гения? Судя по моему опыту, единственный способ выжить в такой обстановке — повышать уровень своих знаний и не давать гению забыть, что окружающие его люди тоже иногда генерируют неплохие идеи. Да, это трудно, но, в конце концов, начальник есть начальник, тут никуда не денешься. Я не призываю вас начинать полномасштабную интеллектуальную войну, но отвечать на любые замечания менеджеров подобного рода чем-нибудь в духе «Да, сэр!», поверьте мне, не стоит. Допустимый ответ на их предложения мог бы звучать так (в уважительном тоне): «Да, хорошо, но не думали ли вы о... [далее следует стройная аргументация]?» Не пытайтесь препираться с гением, если по своим познаниям в предмете спора вы ему заметно уступаете, — уважения к вам от этого очевидно не прибавится.

Помнится, мне встречались своеобразные тандемы, состоящие из гения и его помощника по административным вопросам, которому удавалось компенсировать управленческие недостатки шефа. Сочетание это весьма рискованное, но иногда оно срабатывает. Действительно, интеллектуалы на лидерских позициях в нашей индустрии нужны; при этом не менее важно, чтобы их умственные способности не ограничивались технологическими проблемами и распространялись также на пользователей и исполнителей.

Строители империй тьмы

Лидеров, относящихся к типу строителей империй, можно с равным успехом называть политиканами. Интриги в отношениях между сотрудниками корпораций наблюдаются сплошь и рядом, и если дело касается создания качественных программных продуктов, ни к чему хорошему они не приводят. Строители империй руководствуются в основном одной-единственной целью — подбором преданных исполнителей их воли; при этом очень часто они забывают о необходимости претворять в жизнь цели компании. Поначалу они осыпают подчиненных разного рода поблажками и подарками, но цели построения сплоченной команды перед ними не стоит; они озабочены только одним — превращением своих сотрудников в роботов, беспрекословно исполняющих их предписания. Руководители, не умеющие воспринимать несогласных, не имеют ничего общего с лидерами, — скорее их можно охарактеризовать как коллекционеров чинов и подчиненных. Прислуживание они ценят значительно выше, чем производительность.

Строители империй руководствуются в основном одной-единственной целью — подбором преданных исполнителей их воли; при этом очень часто они забывают о необходимости претворять в жизнь цели компании.

Как-то мне довелось работать в компании, у которой было две группы разработчиков, дислоцированные в разных городах. После нескольких успехов и поражений, случившихся в обеих группах, одна из них стала обнаруживать явные признаки доминирования в деятельности корпорации. Ее руководитель, заручившись

поддержкой других ответственных лиц, сумел обеспечить для своих подчиненных наилучшие условия труда. Все они работали за 21-дюймовыми мониторами, пользовались современными ноутбуками и новейшими моделями карманных компьютеров, имели возможность посещать любые курсы. Помимо прочего, им платили лучше, чем остальным. При любой попытке объединить усилия двух групп между ними неизменно возникали разногласия, что немало расстраивало менеджера, отвечавшего за координацию всех процессов разработки в компании. Все эти годы на разработку любого мало-мальски качественного продукта затрачивались неимоверные усилия — и все из-за распрей между двумя группами.

Империи играют положительную роль только при определенных условиях: во-первых, если в них создаются качественные продукты, во-вторых, если они не замыкаются на решении своих узкокорыстных задач. По моему мнению, компании Microsoft удалось привнести в нашу индустрию чувство единения. Быть может, относительно Microsoft (если хотите, пишите это название по-другому: «Micro$oft») у вас другие соображения, но, как мне кажется, эта корпорация пришла к доминирующему положению на рынке продуктов для персональных компьютеров и сделала из своих продуктов стандарт «де-факто» исключительно собственными усилиями. Империи тьмы, напротив, лишь удовлетворяют личные амбиции своих руководителей, а клиентов воспринимают как одно из средств достижения целей; основная цель же, как правило, заключается в расширении влияния любыми средствами. Даже если вы категорически не согласны с моим мнением относительно Microsoft, нельзя не выделить в личностях строителей империй тьмы следующих качеств.

- Все крутится вокруг лидера. Для него крайне важна личная власть и престиж.
- Если вы — среди фаворитов лидера, права на ошибку у вас нет.
- Лояльность ценится выше, чем производительность.
- Вознаграждения (деньги, инструментарий и т. д.) не всегда выдаются исходя из заслуг и часто служат средством воздействия.
- Лидер мыслит в категориях «мы — они», наделяя эту дихотомию враждебным оттенком; все, что так или иначе ставит под угрозу власть лидера, подвергается яростным гонениям.

Какую империю строите вы? Она удовлетворяет потребности клиентов или ваши собственные амбиции? Поймите меня правильно: амбиции — вещь хорошая, и личные устремления руководителей ни в коем случае нельзя сбрасывать со счетов. Более того, амбиции часто стимулируют деятельность руководителя и его подчиненных. В то же время в рамках империи следует избегать тенденции, согласно которой группа разработчиков преследует своекорыстные интересы и пытается наращивать свое влияние за счет стратегических интересов компании в целом.

Если вам довелось работать под началом строителя империи, средств смягчить последствия издержек этого положения (на случай, если они себя проявят) у вас не так уж много. Лучшее, что вы можете сделать, — работать честно. Это качество способно уберечь вас в ситуации, когда над вами начнут сгущаться тучи. Высказывая такое мнение, я, естественно, исхожу из того, что представители верхних эшелонов руководства неусыпно следят за соблюдением интересов компании

и не готовы слишком долго терпеть действия мелких «божков». В то же время следует иметь в виду, что если строитель империи на практике демонстрирует положительные результаты, значит, скорее всего, он устраивает высшее руководство. Да, наш мир несовершенен; в истории много прецедентов длительного существования империй тьмы. Если такая структура выполняет полезную функцию и приносит прибыль, то и в корпоративном мире она имеет право на существование.

Деятельность по строительству империй идет вразрез с курсом на выработку консенсуса в технических вопросах. Лидер, излишне обеспокоенный своим положением, вряд ли потерпит ситуацию плюрализма мнений. Как известно, консенсус вырабатывается в результате конкуренции идей; с этой точки зрения мы можем сделать вывод о том, что технологическая база империй зачастую страдает. В процессе формирования лидерского стиля вы должны иметь это в виду. Вы ведь хотите стать сильным лидером, не так ли? Сила предполагает способность выдерживать стрессовые ситуации и не сдаваться под грузом обстоятельств. С давлением, которое оказывает на нас быстрая смена технологий и бизнес-требований, сложно справиться без такого качества, как гибкость. Но гибкость и строительство империй — вещи не слишком сочетаемые. Стараясь наращивать производительность своих подчиненных (как в количественном, так и в качественном отношениях), не забывайте об этом.

Деятельность по строительству империй идет вразрез с курсом на выработку консенсуса в технических вопросах. Лидер, излишне обеспокоенный своим положением, вряд ли потерпит ситуацию плюрализма мнений.

Заигрывание с дьяволом

Уверяю вас: если не обращать внимания на определенные признаки, свернуть на порочный путь очень просто. Проработав над проектом шесть недель без перерыва и выходных, удерживая в голове все его детали, немудрено в какой-то момент послать всех подальше. Такая реакция вполне объяснима — ведь вы же человек, а не робот, в конце концов! Тем не менее на такие вещи нужно обращать внимание. В современных условиях сочетать рабочие обязанности со всеми остальными составляющими повседневной жизни становится все сложнее. Постоянно сталкиваясь с разного рода соблазнами, трудно им не поддаться. Не стоит играть с огнем. Я перечислю ряд признаков, при появлении которых стоит насторожиться.

Вы достигли своего предела

Вы работаете все больше, но эффективность продолжает падать. Отношения с подчиненными стремительно ухудшаются; у вас регулярно случаются приступы ярости, и вы все чаще прибегаете к необоснованной критике. Обзоры кода проводятся все реже, а недостаток концентрации не позволяет вам вырабатывать конструк-

тивные и полезные предложения. Ваше ощущение подавленности передается окружающим, продуктивность сотрудников падает.

Пора остановиться и дать знать всем остальным участникам проекта, что вы выжимаете из себя максимум возможного и срыва сроков сдачи не избежать. Это сложная задача, но по достижении предела продуктивности у вас просто не остается другого выхода. Попытки превзойти свои физические возможности приведут лишь к более плачевным последствиям.

Вы прыгнули выше головы

Работа вас достала. Вы обнаруживаете неспособность разбираться в сложных вопросах с необходимой оперативностью и теряете контроль над персоналом. Вы с неохотой ездите на работу, ищете любую возможность остаться дома, мечтаете о том, чтобы сменить сферу деятельности. Избегая принятия решений, вы надеетесь, что ситуация разрешится сама собой.

Посоветуйтесь с другими руководителями. Обратитесь за поддержкой к начальнику — в конце концов, он для этого и существует. Если возможно, сбавьте темп работы и обсудите сложившуюся ситуацию с подчиненными — возможно, они подскажут какие-то стратегии и тактики, которые позволят вам вернуть себе контроль над процессом разработки. Свои недочеты лучше признавать как можно раньше — в противном случае ближе к завершению проекта ваши ошибки все равно станут для всех очевидными. Если вернуть себе контроль не удастся, возможно, пора уходить. Только вы сами можете определить, насколько вы подходите для выполнения лидерских функций. В таких вещах лучше признаваться самому себе, пока за вас это решение не приняли окружающие.

Вас бесит критика

Приступы ярости, как правило, свидетельствуют об игнорировании двух предыдущих признаков. Ярость, по сути, — это неосознанная реакция на чувство бессилия или на неудачу. На этот признак следует обращать особое внимание: ярость может подорвать ваш авторитет и восстановить его впоследствии будет очень сложно. Критика, в лучшем смысле этого слова, помогает повысить производительность. Раздражает она в том случае, если вы убеждены, что сделали все возможное.

Если вы работаете с полной самоотдачей на протяжении многих недель или месяцев, достигать высот продуктивности становится все сложнее. Прибавьте к этому недостаток опыта и несоответствие личностных характеристик занимаемой должности, и вы превратитесь в бомбу замедленного действия. Собрать все кусочки после взрыва очень трудно, так же как, например, переделать все экземпляры скомпилированного и распространенного исполняемого файла.

Как выжить в период упадка и восстать из пепла

У этой книги практическая направленность. В теории, если следовать разумным советам (а лучшие советы, несомненно, мои), можно избежать опасностей,

способных подорвать ваше лидерство. В то же время реальность сильно отличается от теории. Каждый лидер может указать на огрехи в своей деятельности, многие из которых я упомянул в этой главе. Таким образом, навыки преодоления трудностей и возобновления успешной деятельности совершенно необходимы.

Какие отношения нарушились за время ваших трудностей? Можно ли поправить дело? Что делать с неудачными проектами и пропущенными сроками сдачи? Как компенсировать ущерб? Если вам предстоит восстановить свои пошатнувшиеся лидерские позиции, вы должны обязательно задать эти вопросы.

Начните с людей. Найдите тех, кто пострадал из-за допущенных вами ошибок, и извинитесь перед ними. Не ищите себе оправданий — признайте свою неправоту. Не вдавайтесь в сентиментальность или драматизм — просто дайте знать, что вы сожалеете о том, что случилось, и двигайтесь дальше. Незаслуженно пострадавшие ждут, когда перед ними извинятся, и если вы признаете свои ошибки, вас будут уважать за честность. Дело не том, чтобы установить теплые отношения, а в том, чтобы восстановить фундамент лидерства. Необходимость ставить точки над «i» в ваших отношениях с персоналом не слишком приятна, но без этого не обойтись. Если вы ранее успели проявить лидерские качества, сотрудники вас простят. Не обманывайте их ожидания — не извиняйтесь по электронной почте. Если обида носит личный характер, то и извиняться надо без посредников — в конце концов, никто не заставляет вас документировать свои огрехи.

Ущерб, нанесенный процессам, не носит столь личный характер, но компенсировать его не проще, чем испорченные человеческие отношения. Если допущенные вами ошибки явили собой прямое нарушение правил компании, принятых принципов работы или методологии разработки, лучше всего постараться обеспечить соответствие им. Составьте для себя план повторного изучения всех значимых правил компании, имеющих отношение к вашей деятельности. Правила имеют одну четкую функцию — они обеспечивают единообразие в работе и в достижении качественных результатов. Мы, разработчики, очень ревностно относимся к своей независимости, в связи с чем часто легкомысленно воспринимаем правила. Вы — руководитель и лидер, — демонстрируя пример несерьезного отношения к правилам, можете ожидать того же от своих подчиненных.

Как избежать заката

В идеале лидер вообще не должен попадать в ситуации заката. Чтобы восстановить лидерские позиции, вы должны придерживаться фундаментальных принципов лидерства — о них, кстати, речь пойдет в следующей главе. Обзор негативных аспектов деятельности руководителя, приведенный в этой главе, был призван помочь вам разобраться в некоторых стереотипных стилях руководства, которые за незнанием лучшего мы иногда принимаем или имитируем. Приходя на руководящие посты из рядов программистов, мы, как правило, лишены формального образования в области менеджмента и лидерства, а потому имеем обыкновение переносить все наработанные нами в качестве программистов админи-

стративные навыки на свою новую роль. Некоторые из навыков действительно применимы к руководящей деятельности, но далеко не все.

Резюмирует все вышесказанное табл. 7.3 — надеюсь, она поможет вам избежать негативных аспектов лидерства.

Таблица 7.3. Порочные стили руководства

Стиль	Причины и симптомы	Последствия	Рецепт
Мелочная опека	Ориентация на собственные силы, неверие в способности подчиненных, страх совершить ошибку	*Для менеджера* усталость и раздражение; *для сотрудников* отсутствие чувства сопричастности к созданию продукта, ощущение ненужности и бессилия; *для продукта* исключительная роль менеджера в его создании осложняет его сопровождение	Переориентация на окружающих, выстраивание доверительных отношений с сотрудниками, борьба со страхами путем сотрудничества с персоналом
Неорганизованность	Откладывание проблем на потом, отвлечение от рабочих вопросов, неопытность	*Для менеджера* огромное количество отложенных решений; *для сотрудников* хаос, утеря координации (все делают то, что считают нужным); *для продукта* срыв сроков сдачи, несоответствие спецификациям	Концентрация на текущих задачах, самоорганизация, помощь со стороны других руководителей
Неверное приложение гением своих сил	Пренебрежение вопросами кадрового обеспечения, чрезмерное внимание к технологии	*Для менеджера* недоумение по поводу незаинтересованности персонала в реализации задумок; *для сотрудников* боязнь оспаривать технические предложения начальника; *для продукта* сильно осложняется сопровождение из-за реализации в продукте предложений, исходящих, прежде всего, от менеджера	Сотрудничество с помощником по административным вопросам или обучение основам лидерства и менеджмента
Строительство империй	Основное внимание уделяется укреплению своих позиций и власти в сочетании с политиканством	*Для менеджера* постоянная неудовлетворенность, стремление расширить империю; *для сотрудников* изоляция и трения с другими подразделениями компании; *для продукта* сотрудничество в принятии решений по продукту заменяется принуждением	Обеспечение для сотрудников возможности проявить себя и отказ от политиканства

Как видно из табл. 7.3, употребление порочных стилей руководства приводит к определенным последствиям и для самого менеджера, и для его подчиненных, и для продукта. Представители этих стилей не уделяют внимание вопросам лидерства — они либо ведут своих сотрудников в неверном направлении, либо вообще их никуда не ведут. Знание последствий применения упомянутых

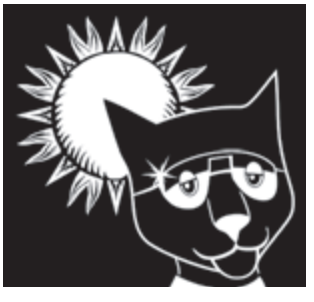
стилей руководства поможет вам побороть их симптомы. Впрочем, в каждом конкретном случае вы должны проанализировать личные причины появления признаков того или иного стиля — ведь если лечение не затрагивает первопричины болезни, уничтожить ее оно не сможет.

Что дальше

В следующей главе мы рассмотрим фундаментальные качества лидера. Получить адекватное представление о полноценном лидерстве можно, сравнив стили, описанные в этой и следующей главах. Быть может, знание отрицательных аспектов лидерства при одновременном рассмотрении его положительных качеств приведет вас к прозрению. Всем нам время от времени нужны источники вдохновения, и, я надеюсь, материал следующей главы поможет вам обрести уверенность в своих силах и дополнительные навыки лидерства.

Глава 8

Восход лидера



Учитывая, что в предыдущей главе мы рассмотрели негативные аспекты лидерства, полагаю, оно заслуживает реабилитации. Знание всех проблем, с которыми время от времени сталкиваются лидеры, заставляет усомниться в том, можно ли вообще преуспеть в этом качестве, — уверяю вас, можно. Хороший лидер — это гораздо больше чем блестящий руководитель. Лидерство, хотя оно и строится на основе различных стилей руководства, подразумевает приложение значительных усилий, которые отнюдь не ограничиваются координацией людей и процессов. Лидеры должны быть на передовой — они призваны разрабатывать новые пути развития компании, привлекая для этой цели надлежащие технологии. Не всегда лидеру удается с первой попытки заставить окружающих двигаться в выбранном направлении. Мобилизовать сотрудников на решение определенных задач помогает харизма — одна из черт лидера, которую, по моему мнению, практически невозможно привить. Харизматический лидер тем более должен понимать, в каком направлении двигаться.

Подобно программному обеспечению, лидерство выстраивается на определенном фундаменте. Для программных продуктов в этой роли выступает архитектура, а для лидерства — ваш характер. Характер, который должен отвечать четким требованиям. Характер, предполагающий ответственность за подготовку технических достижений компании и за их реализацию силами ее сотрудников. Подробнее об этих фундаментальных принципах мы и будем говорить в настоящей главе — культивируя и дополняя их, вы сможете добиться в своей деятельности грандиозных успехов.

Подобно программному обеспечению, лидерство выстраивается на определенном фундаменте. Для программных продуктов в этой роли выступает архитектура, для лидерства — ваш характер.

Фундаментальные принципы лидерства

Начну с вопроса, который, по-моему, ничем не хуже дилеммы курицы и яйца: кто появился раньше — лидер или его последователи? Не знаете? То-то — это вопрос с подвохом (это я так подготавливаю вас к более глубокому анализу фундаментальных принципов). Я думаю так: чтобы лидера можно было назвать лидером, ему нужны последователи; в то же время хороший лидер сам формирует для себя группу последователей. Процесс подготовки последователей прекрасно иллюстрируют пять основных принципов, которые, как мне кажется, определяют суть характера любого лидера. На рис. 8.1 эти принципы приводятся в двух вариантах группировки. Первый вариант иллюстрирует очередность применения принципов, второй — их цикличность.

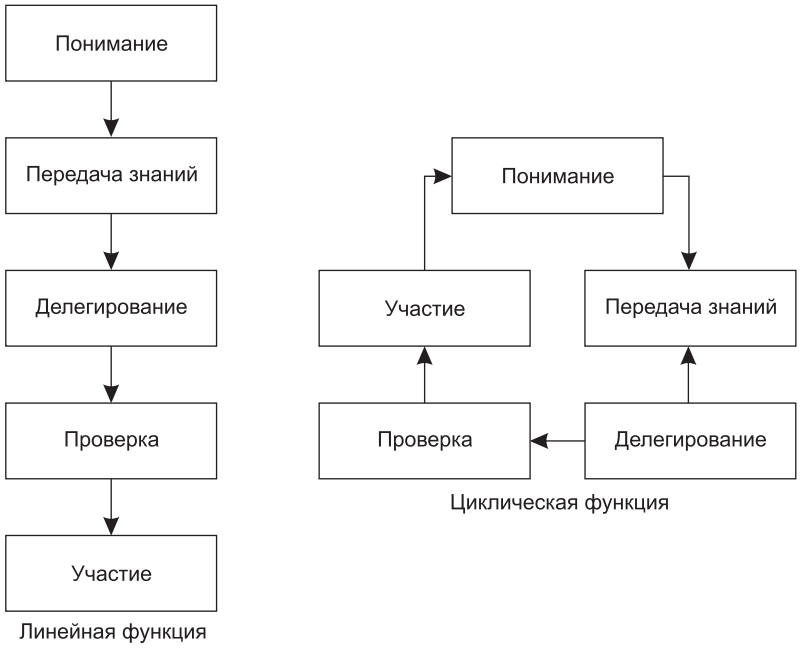


Рис. 8.1. Основные принципы лидерства

Каждый из принципов важен как по своей сути, так и с точки зрения взаимодействия с остальными. Упомянутые принципы лидерства вы должны усвоить полностью — ведь они подобны фундаменту здания, который по определению должен быть его самым надежным элементом.

Понимание

Прежде чем приступать к обязанностям лидера, вы должны четко уяснить, в каком направлении поведете своих подчиненных. Для того чтобы забраться на вершину Эвереста, одной лишь карты недостаточно. Для этого нужно изучить действия тех, кому удалось это сделать до вас и кто после этого умудрился вернуться живым. Вы должны знать условия восхождения, связанные с ним риски, препят-

ствия, встречающиеся на пути, правила применения соответствующего оборудования, цель путешествия. Правда, нередко энтузиасты совершают восхождение на гору только лишь «потому, что она есть». Полагаю, впрочем, что на самом деле цель заключается в другом — люди стремятся покорить то, что считают могущественнее себя; аналогичным образом для создания конкурентных преимуществ и завоевания доли рынка коммерческие предприятия производят товары и услуги. Быть может, такая аналогия слишком груба, но рынок, по моему мнению, представляет собой арену жесткой конкуренции и битв не на жизнь, а на смерть; и наша конечная цель как разработчиков программных средств заключается в том, чтобы выпустить продукт (или услугу), который может помочь нашей компании выиграть несколько таких битв. Битвы на рынке способны выигрывать только лидеры, понимающие характер, методы и цели военных действий в области программного обеспечения.

Битвы на рынке способны выигрывать только лидеры, понимающие характер, методы и цели военных действий в области программного обеспечения.

Вы должны полностью уяснить для себя цели компании — лишь в этом случае вы сможете донести их до своих подчиненных. Естественно, процесс изучения начинается с перспективного представления, хотя им одним задача не исчерпывается — специалистам в области разработки программных средств приходится копать довольно глубоко. Одно из обязательных качеств лидера — умение отличать лес от деревьев. Мы, программисты, нередко блуждаем среди деревьев, не понимая топологии леса. Лидер программистов не может себе позволить заблудиться — слишком серьезными могут быть последствия и для подчиненных, и для продукта.

Пока вы не обдумаете, не взвесите и не обоснуете каждое коммерческое требование, добравшись, наконец, до «эврики»[1], на комплексное понимание проблемы не рассчитывайте. Наивно надеяться на то, что сотрудники, прочитав спецификацию продукта, все сразу поймут и создадут реализацию именно такой, как требуется. Только выстроив для себя подробный план реализации от начала до конца, вы сможете руководить процессом и отслеживать его. Здесь опять же необходимы время и настойчивость. Первого никогда не бывает в достатке (за это я ручаюсь!), а второе условие напрямую зависит от того, насколько серьезно вы относитесь к своим лидерским обязанностям. Что касается времени, то у всех нас его поровну — просто некоторые им пользуются рациональнее, чем другие. Относительно того, как опасно пренебрегать личной ответственностью за деятельность подчиненных и судьбу продукта, лучшие рекомендации дает опыт.

Требования кто-то формулирует — значит, очевидно, они поддаются пониманию. Это своеобразная трактовка известного высказывания о том, что если есть воля, значит, есть и выход — правда, «волю» здесь следует понимать как исследование вопроса вплоть до его полного понимания. План анализа проблемы можно изложить следующим образом.

---

[1] Именно так (в переводе с греческого: «нашел!») выразился Архимед, открыв основанный на принципе плавучести метод проверки чистоты золота.

1. Прочтите требования дважды: один раз, чтобы понять широту замысла, второй — чтобы постичь его глубину.
2. Сопоставьте требования с известными методиками реализации и выявите те части функциональности, для реализации которых потребуются новые разработки.
3. Набросайте предварительный план макетирования проектов — он поможет выявить среди требуемых свойств неизвестные.
4. Составьте на основе документации с требованиями контрольный список, который, в свою очередь, послужит исходной точкой для формулирования задач. Так вы сможете привязать каждое требование к набору задач и тем самым гарантировать реализацию свойства.

Понимание рождает решения. Соответственно переходим к следующей роли лидера — передаче знаний коллегам.

Передача знаний

Слово «благовещение» я первый раз услышал в церкви еще ребенком. В религиозном контексте оно употребляется в совершенно четком смысле и обозначает распространение хорошей новости. Много позже я услышал этот термин в светской интерпретации от Microsoft — словосочетание «благовестник продукта»[1] показалось мне чрезвычайно точной характеристикой для восторженного молоденького преподавателя, к которому я ходил на курсы программирования. Способность передавать знания есть второй необходимый признак лидера. Иногда говорят: «тот, кто умеет, делает; тот, кто не умеет, преподает». Это изречение я считаю неверным и предлагаю встречную мысль: у того, кто не может адекватно изложить свои мысли, их слишком мало. Полагаю, что именно эта проблема обусловливает плохую передачу информации в бизнесе: недостаточное понимание проблемы порождает комплекс, в результате чего субъект, который, по смыслу, должен эту проблему доходчиво изложить, начинает надеяться на то, что окружающие смогут интуитивно разобраться в ней и уяснить свои задачи. Наитие выходит на первый план, если документацию с бизнес-требованиями по ночам не читать, а использовать по естественному назначению — в качестве подушки; впрочем, и наитие никогда не сможет заменить четкое изложение мысли.

Цель передачи знаний — обеспечить понимание персоналом предъявленных к продукту требований на том же уровне, на котором их понимает лидер. Итак, каким образом вам самому удалось понять проблему в комплексе? Восстановите последовательность действий, направленных на изучение требований, и перенесите ее на процесс обучения сотрудников. Тот, кто способен четко доносить свои знания до окружающих, сможет преуспеть в педагогике. Да, действительно, даже самый лучший учитель не может обойтись без заинтересованных учеников, но как лидер программистов вы располагаете в этом отношении существенным преимуществом — ваши «студенты» работают под вашим же началом, и никуда им от

---

[1] В оригинале — «product evangelist». Если бы не параллель с евангелистской терминологией, которую приводит автор, мы, несомненно, предпочли бы перевести это словосочетание как «проповедник продукта». — *Примеч. перев.*

вас не деться. Может быть, они не всегда вас слушают, но ведь вы — шеф и поэтому располагаете методами принуждения к обучению. Поощрение, несомненно, выигрывает в сравнении с принуждением, но бывают ситуации, когда выбирать не приходится. Вернемся к основной мысли: если передача знаний равнозначна педагогической деятельности, то как лучше составить «план урока»? Элементы, приведенные в табл. 8.1, являются минимально необходимыми для адекватной передачи знаний.

Тот, кто способен четко доносить свои знания до окружающих, сможет преуспеть в педагогике.

Таблица 8.1. Необходимые элементы эффективной передачи знаний

Формальные элементы передачи знаний	**Функциональные элементы передачи знаний**	**Стилистические элементы передачи знаний**
Введение в проблему. Общее ознакомление с проблемой, определение рамок и плана изложения	*Объяснение*. Пояснение технической терминологии и упрощенное объяснение понятий	*Голос*. В зависимости от характера аудитории, места проведения занятия и степени срочности объяснения материала варьируется от формального до неформального
Обсуждение. Описание проблемы в деталях, разделение проблемы на отдельные темы, стремление к самодостаточности каждой темы и одновременно к ее связности с другими темами	*Аргументация*. Убеждение с позиций логики и точности языковыми средствами	*Тон*. Серьезный, но не слишком; подходящий для конкретной бизнес-ситуации. Юмор допускается и приветствуется
Переход. Связывание всех элементов изложения воедино и демонстрация установленных между ними связей	*Демонстрация прикладных аспектов проблемы*. Переход от теории к практике и реализации предмета изложения	*Ритм*. Закрепить отдельные элементы пройденного материала помогают голосовые колебания и изменение тона
Заключение. Логика передачи знаний должна стать основой для построения плана действий	*Иллюстрации*. Аналогии и метафоры, помогающие показать альтернативные аспекты проблемы и тем самым углубить ее усвоение	

Многие лидеры излишне увлекаются стилем — в основном по той простой причине, что у них не все в порядке с содержанием. Основное внимание нужно все-таки уделять содержанию, ну а тонкости изложения можно будет наработать с опытом. Перечисленные в табл. 8.1 элементы передачи знаний следует задействовать как при устном, так и при письменном общении.

Необходимо различать запланированный процесс передачи знаний и случайные разговоры на ту или иную тему. Поскольку вы — лидер, ваши замечания, пусть даже высказанные по случаю, зачастую воспринимаются сотрудниками как официальные распоряжения. Большую часть жизни мы доносим до окружающих информацию

интуитивно, не пользуясь заготовками и отталкиваясь исключительно от текущего контекста. Различия между бессистемным и формальным мышлением следует обязательно иметь в виду; бывают ситуации, когда случайно высказанная мысль приводит к непредусмотренным разрушительным последствиям для бизнеса. Способность контролировать поле боя общения помогает выиграть битву понимания.

Способность контролировать поле боя общения помогает выиграть битву понимания.

Как лучше всего донести до программиста суть требований? Показать ему частично функционирующий макет. Даже одни пользовательские интерфейсы способны помочь программисту составить представление о задаче. Впрочем, не стоит сбрасывать со счетов архитектуру. Как я говорил в главе 6, прежде чем собирать урожай программных объектов, необходимо разбить сад, в котором вы намерены их выращивать.

Испытанный способ улучшить передачу знаний — обратиться к помощи UML, PowerPoint или Visio. Возможно, в вашей организации применяется оригинальный процесс документирования требований и составления проектных документов. Если это так, придерживайтесь его, а при необходимости адаптируйте к конкретному проекту. Впрочем, имейте в виду, что строить разработку программных средств исключительно на основе документов не стоит. Нередко такие важные элементы процесса передачи знаний, как макетирование и критический обзор предварительных аспектов реализации, просто-напросто игнорируются. Полагаю, что, став руководителем и лидером, вы начали испытывать ностальгию по кодированию. Если так, то, обратив внимание на два упомянутых элемента, вы сможете освежить навыки программирования и одновременно решить свою непосредственную задачу. Утверждения о «самодокументированности кода» слышны повсюду, но на самом деле они редко соответствуют действительности. Конструирование макетов как средств передачи знаний приносит пользу в двух отношениях: во-первых, удовлетворяет привычку к кодированию, во-вторых, помогает донести до окружающих ваше понимание решения поставленной задачи.

Делегирование

Объяснив подчиненным, какая задача перед ними стоит, заставьте их ее решить. Делегирование — это тоже искусство, а о том, почему оно является вашей главной обязанностью, я уже неоднократно говорил. Способность к делегированию — это одно из тех качеств, которое вы, лидер, должны постоянно совершенствовать. Косвенно этой темы я касался в главе 1. Сопоставив перечисленные в ней типы программистов с личностными качествами ваших сотрудников, вы сможете принять правильное решение относительно распределения задач между ними.

В идеале проект нужно разбить на рабочие единицы, которые впоследствии можно будет распределить между всеми разработчиками. Впрочем, на практике этот подход не всегда себя оправдывает. Может статься, что для решения отдельных задач по проекту или даже для комплексной его реализации лучше всего подходит ограниченная группа сотрудников. Иногда, в тех ситуациях, когда две го-

ловы лучше, чем одна, программистов имеет смысл разбивать по парам. Этот прием описан в методике разработки программных средств под названием экстремального программирования[1]. В рамках этой методики также культивируется другой тип делегирования — он предусматривает создание команды из двух программистов, в которой один пишет код, а другой ежедневно тестирует его и проводит критические обзоры. Если подобрать подходящих кандидатов (лучше всего — опытного программиста и новичка), эта схема делегирования оказывается очень эффективной — сам видел. Однако если подобрать партнеров неверно, присущая программистам независимость сведет эффективность этой замечательной идеи нулю. Помимо прочих навыков делегирования, вы должны понимать, какие из ваших сотрудников могут работать вместе, а какие — нет.

Не забывайте и о хрестоматийном правиле, согласно которому введение в проект новых сотрудников на завершающих стадиях разработки только оттягивает срок его сдачи. Делегировать можно задачи, но не темпы их решения. Если вы обнаруживаете, что разбить проект на отдельные задания трудно, значит, возможно, проблема кроется в проектном решении. Вероятно, плану конструирования недостает модульности. Иногда в самом начале работы над проектом кажется, что никаких проблем с делегированием не возникнет. Впоследствии, однако, выясняется, что одни группы уже решили поставленные перед ними задачи, а другие запаздывают. Подобные ситуации возникают из-за неверного представления о продолжительности выполнения тех или иных заданий. Эта проблема опять же связана с проектированием. Качественное проектное решение (при наличии достойной архитектуры) обеспечивает удобство делегирования по модулям.

Не забывайте и о хрестоматийном правиле, согласно которому введение в проект новых сотрудников на завершающих стадиях разработки только оттягивает срок его сдачи. Делегировать можно задачи, но не темпы их решения.

Нужно уметь корректировать план делегирования на ходу. К необходимости его корректировки нужно быть готовым с самого начала — она не должна стать для вас сюрпризом. В ходе выполнения проекта от начальных стадий к завершающим у лидера нередко появляется ощущение стрельбы по движущейся мишени — вот почему для него так важен опыт. Если бы лидеры разработки программных продуктов могли тренироваться в стрельбе по движущимся мишеням на стрельбище, было бы просто здорово. К сожалению, практиковаться нам приходится в реальных условиях — иначе вряд ли мы срывали бы сроки. Чем больше вы будете тренироваться, тем совершеннее станут ваши навыки. Делегирование — это лучший помощник в деле стрельбы по мишеням. Стоит вам попытаться сбить их все самостоятельно (то есть поддаться соблазну мелочной опеки), патроны кончатся раньше времени.

Проверка

В этом контексте на авансцену выходит знакомый принцип «ожидай, но проверяй». Естественная задача, возникающая после делегирования, заключается в контроле

[1] Среди изданий, перечисленных в библиографии, присутствует книга «Extreme Programming Explained» Кента Бека (Kent Beck).

за ходом выполнения заданий. Это не мелочная опека — это именно проверка. Лучший способ приблизить перспективу завершения проекта в срок — проводить регулярные сборки компонентов вплоть до полной готовности кода. Такие действия, помимо прочего, помогают избежать попадания в проектный тупик. Слишком много времени теряется, если на подходе к завершению цикла разработки приходится начинать все сначала — по той лишь причине, что проектное решение себя не оправдало. Одним из элементов вашей повседневной деятельности должна стать ежедневная проверка результатов работы сотрудников; при приближении сроков сдачи такие проверки нужно проводить еще чаще.

Эффективный мониторинг подобен непрекращающемуся критическому обзору кода — он также предполагает контроль за соответствием проектному решению и стандартам кодирования. Чем дольше вы будете затягивать проведение обзора, тем больше задач вам придется решать, — а если откладывать процесс слишком долго, у вас просто не останется времени на то, чтобы разобраться в запутанном коде.

Перечислю некоторые методики проверки.

- Ежедневно наведывайтесь к своим подчиненным и интересуйтесь, как у них идут дела. Если личный контакт не представляется возможным, воспользуйтесь телефоном или, на крайний случай, электронной почтой.
- С регулярностью в несколько дней тестируйте готовые модули кода. Эта деятельность роднит вас с тестерами, но вы должны воспринимать ее как свою первоочередную обязанность.
- Регулярно пробуйте интегрировать компоненты в целостную систему и проверяйте архитектуру на стройность. Здесь придется «поиграть» с кодом. Так вы сможете выявлять ошибки, допущенные при конструировании, и держаться в курсе методов, которыми ваши программисты решают поставленные перед ними задачи.
- Проводя еженедельные совещания сотрудников (о них мы говорили в главе 5), не забывайте отслеживать состояние проекта и вырабатывать вместе с подчиненными новые идеи. Эта деятельность дополняет ежедневные визиты к сотрудникам и позволяет привлечь к решению проблем общего характера коллективный разум.
- Тех участников группы разработчиков, которым удалось быстро расправиться со своими задачами, следует привлекать к тестированию результатов труда остальных. Делать это нужно с осторожностью, допускать высказывания типа «тестирование — это не мое дело» нельзя. Программирование не ограничивается написанием кода, оно также предполагает обеспечение его реального функционирования.
- Требуйте от программистов документировать все новые модули кода и составляйте из этих отчетов сопроводительную документацию для отдела тестирования. Тем самым вы дадите возможность себе и своим подчиненным оценивать ход процесса и соответствие заданным требованиям.

Допускать высказывания типа «тестирование — это не мое дело» нельзя. Программирование не ограничивается написанием кода, оно также предполагает обеспечение его реального функционирования.

Вышеупомянутые методики (впрочем, их список можно расширять) помогают превратить процесс мониторинга в продуктивную и конструктивную деятельность, не ограничивающуюся оголтелой критикой. Если лидер не будет проверять своих подчиненных, кто-нибудь из них может выпасть из общего темпа разработки, вследствие чего продукт не будет сдан вовремя. Достичь сопоставимых успехов в разработке должны все участники группы — в противном случае толку от такой деятельности будет очень мало.

Участие

Несмотря на свое положение лидера, вам нередко придется участвовать в процессе кодирования. Если сочетать эту деятельность с делегированием и проверкой, ничего страшного не произойдет. Участие в «грязной работе» поможет вам утвердиться в роли лидера, хотя здесь нужно стараться воздерживаться от мелочной опеки. Разделяя обязанности по кодированию со своими подчиненными, вы не раз получите от них одобрение; впрочем, имейте в виду, что увлекаться кодированием для менеджера опасно. С учетом численности персонала отдела эта деятельность может стать необходимостью, однако за написание кода, с одной стороны, и организацию этого процесса, с другой, отвечают разные участки мозга. Переключаться с одного вида деятельности на другой довольно сложно. Для ведения административных дел далеко не всегда требуется такая же концентрация, как для программирования; таким образом, исполняя обязанности руководителя, можно позволить себе отвлечься на телефонный звонок или письмо, пришедшее по электронной почте. Что же касается кодирования, полагаю, мне не нужно вам объяснять, насколько губительными оказываются разного рода раздражители.

Если бы у нас вместо мозгов были микропроцессоры, в которых при «прерывании» предусматривается сохранение состояния системы, параллельное исполнение многочисленных заданий не составляло бы для нас такой проблемы. Мы можем сколь угодно хвастаться способностью делать несколько дел одновременно, однако, как правило, качество исполнения каждого из них оставляет желать лучшего. В нашей работе необходима концентрация — любой специалист в области разработки программных средств должен стремиться в каждый конкретный момент заниматься чем-то одним.

Как я уже говорил в разделе «Проверка», ваше участие в кодировании зачастую сводится к тестированию. Это жизненный пример участия лидера в процессе. Трудно найти компанию, в которой обязанности по интеграции и тестированию (до наступления этапа альфа-тестирования) мог бы взять на себя кто-то помимо лидера группы разработчиков. Готовьтесь к нетривиальным испытаниям ваших технических навыков и учитесь рационально использовать время. Справившись с этими трудностями, вы сможете укрепиться в положении лидера. Если вы будете активно участвовать в разработке, вам вряд ли хватит стандартных сорока часов в неделю. Правда, здесь вас подстерегают опасности — вспомните ощущения, которые, как я говорил в предыдущей главе, можно охарактеризовать одной фразой: «Мне уже все равно». Если человек доходит до такого состояния, то, наверное, менять что-либо уже поздно. Вы должны постоянно следить за уровнем своей загруженности — с тем, чтобы не выпасть из процесса в самый неподходящий момент. Как помните, разработка программных продуктов — это не спринт, а марафон; в этом контексте умение взять правильный темп есть необходимое

условие стабильной работы всей команды. Чтобы выиграть марафон, нужно поддерживать взятый темп, прилагать усилия к тому, чтобы переставлять ноги, следить за рельефом и не забывать, как это здорово — пересечь финишную линию с достойным временем.

В школе и колледже я занимался бегом по пересеченной местности. Через несколько лет, за которые я успел жениться, отслужить в армии, завести детей и устремиться к достижению американской мечты, тренировки ушли на второй план. Много позже, пропустив несколько десятилетий, я начал тренироваться заново и попытался войти в форму. Конечно, сейчас я уже не могу пробежать пять миль за тридцать минут. Пару лет назад у меня это получалось за сорок минут, но, надо сказать, не так легко, как в молодости. Каждый раз, когда я пытаюсь побить собственный рекорд, мне приходится постоянно поддерживать концентрацию. Я обнаружил, что если во время пробежки вслух произносить «сконцентрируйся!», становится легче. Этот режим я стараюсь переносить в свою профессиональную деятельность. Участие — это способность пробежать всю дистанцию, ни разу не теряя внимания. Иногда, поскольку вы лидер, придется исполнять роль тренера, стоящего у трассы, но не стоит забывать — лучшие лидеры сами умеют бегать.

Иногда, поскольку вы лидер, придется исполнять роль тренера, стоящего у трассы, но не стоит забывать — лучшие лидеры сами умеют бегать.

В общем, участие предполагает переживание трудностей и побед вместе с подчиненными. Для этого нужно быть всегда готовым помочь окружающим, уметь прогнозировать возникновение проблем и решать их прежде, чем они окажут воздействие на сотрудников. Другой гранью участия является проявление здорового интереса к личной жизни подчиненных. Не нужно ни к кому навязываться в друзья — просто проявите понимаете того факта, что исполнением рабочих функций их жизнь не ограничивается.

Возвращусь ненадолго к примеру с моими юношескими беговыми упражнениями. Однажды, помнится, наш тренер сказал нам, потным и запыхавшимся, примерно следующее: «Придет время, ребята, и вы будете рассказывать своим детям, как быстро когда-то бегали!» Тогда я не понял, к чему это он сказал. Тридцать восемь лет спустя я уверен: он имел в виду, что участие в команде придает дополнительный смысл индивидуальным стараниям. Вы должны учитывать это обстоятельство в процессе взаимодействия со своими подчиненными — не пренебрегая тренерскими обязанностями, как можно активнее участвуйте в их деятельности.

Надстройка

Разобравшись с фундаментальными принципами, пора переходить к более очевидным вещам. Речь идет о некоторых наиболее значимых видах деятельности, которые естественным образом вытекают из вашего лидерского положения. Если вам они не покажутся столь очевидными, значит, необходимо проверить, насколько полно реализованы основные принципы лидерства. В процессе формирования команды лидер испытывает немало приятных минут — вы же не хотите их упус-

тить, не правда ли? Ей-богу, головной боли в ходе руководства процессом разработки и так хватает. Так не будем пренебрегать возможностью разбавить ее более приятными ощущениями!

Наставничество

Обучая других обучать, мы приумножаем свои усилия и повышаем эффективность. Пусть это будет рабочим определением наставничества. Наставничество помогает решить множество различных задач, и чем больше внимания вы ему уделяете, тем серьезнее отдача. Вы уяснили смысл этого хитрого выражения: «обучать других обучать»? Наставничество — это больше, чем преподавание, поскольку оно предполагает передачу педагогических навыков. Как сказал мне мой отец, в колледже я должен прежде всего научиться находить источники нужной мне информации и правильно с ними обращаться. Он был прав. Наставничество подразумевает обучение самообразованию при одновременной передаче знаний.

Обучая других обучать, мы приумножаем свои усилия и повышаем эффективность.

В академической науке педагогике основную роль играют проблемы эвристики. Как наставник вы должны поощрять стремление своих сотрудников к получению дополнительных знаний и экспериментированию, совершенствуя тем самым способности своих подчиненных к самостоятельному решению задач. Последнее качество в контексте разработки программного обеспечения совершенно необходимо. Охотно верю, что, изучая в школе математику, вы ненавидели слово «задача», но ведь, по существу, вся наша жизнь — эта одна большая задача. Вы должны преобразовать требование в работающий программный продукт, а для этого необходимо установить соответствие между данными в области требований и машинной реализацией, воплотив в нее свой богатый опыт пользователя.

Цель наставничества — воспитать лидеров не хуже самого наставника, а может быть, даже лучше него. Чем больше ответственности за процесс разработки вы сможете привить своим подчиненным, тем серьезней будет та помощь, на которую вы сможете рассчитывать при приближении сроков сдачи. Если это возможно, выделите на наставническую деятельность один из дней рабочей недели. Придерживаться такого графика иногда будет сложно; однако ориентируясь на обучение сотрудников, вы помогаете не только им, но и себе. Подготовка к наставнической деятельности укрепляет ваши собственные знания, формирует более ясное представление о рабочих обязанностях, помогает уточнять задания, которые вы намереваетесь распределить между персоналом.

Характерным примером наставничества может стать построение в реальном времени макета, иллюстрирующего тот или иной метод конструирования программных средств. Быть может, имеет смысл на несколько часов присоединиться к сотруднику, которому попалось особо трудное задание. Это отнюдь не мелочная опека — скорее удачный пример сотрудничества. Наставничество — это сочетание участия и понимания (двух фундаментальных принципов лидерства), направленное на повышение квалификации сотрудников.

Наставник — это не обязательно «старый хрыч». Возраст в данном случае не имеет значения; куда важнее готовность делиться знаниями. Для того чтобы группа смогла добиться выдающихся успехов в разработке программных продуктов, ее сотрудники должны друг другу помогать. Инициатором таких отношений должны выступить вы, главный наставник; надо надеяться, на вашем примере этому научатся остальные.

В зависимости от ситуации вы должны демонстрировать владение двумя видами наставничества: целевым и комплексным. *Целевое наставничество* (targeted mentoring) предполагает оказание сотрудникам вполне конкретной помощи, например в написании процедуры вызова интерфейса прикладного программирования или подпрограммы сортировки — в зависимости от потребности. Такого рода наставничество востребовано всегда. Для *комплексного наставничества* (complete mentoring) нужно нечто большее. Оно требует установления с сотрудниками тесных отношений, построенных на искренности и взаимопонимании (в то же время необходимо отдавать себе отчет, что не со всеми сотрудниками это возможно). Выберите из числа подчиненных самых многообещающих — тех, кого вы хотели бы видеть своими последователями, — и уделите их воспитанию максимум внимания. Делясь своими знаниями, старайтесь укреплять отношения с избранниками. Взять хотя бы художника — он может научить студентов, как работать над эскизами, выстраивать тень и перспективу. Уроки художника посещают многие студенты, и, конечно, они перенимают у него часть знаний. Но лишь немногие, лучшие ученики сознательно подготавливаются художником для того, чтобы впоследствии занять его место. Я говорю о роли ученика, подмастерья, адепта. Если вы незаменимы, вас никогда не заменят. Учитывая планы карьерного роста, ничего хорошего в такой незаменимости нет.

Вознаграждение

Выдающиеся успехи нужно поощрять. Вы как лидер должны принимать решения о том, когда и в каком виде выдавать призы. Выбор широк: от повышения заработной платы до похода с отличившимся сотрудником в хороший ресторан или на какое-нибудь спортивное зрелище. Награда должна соответствовать достижениям и основываться исключительно на заслугах. За то чтобы сотрудники выкладывались на все сто, им платят: дополнительные поощрения должны распространяться лишь на тех, кто, превысив ожидания, прыгнул выше головы.

Принципы вознаграждения сотрудников обычно регулируются корпоративной политикой, что, однако, не исключает творческого подхода к этой задаче. Можно, например, провести в период бета-тестирования конкурс, в котором сотрудники будут соревноваться в количестве устраненных ошибок; победителю в таком соревновании, естественно, полагается денежный приз. Оценивать результаты вы должны исходя из характера ошибок и информации о лицах, их допустивших. Ниже я привожу пример схемы подобного рода оценки.

1. Количество исправленных ошибок.
2. Сложность каждой ошибки.
3. Время, потраченное на исправление каждой ошибки.
4. Успешность предложенного решения (зависит от результатов повторного тестирования).

5. Усилия по взаимодействию с другими сотрудниками, направленному на поиск решения (в случае, если такое взаимодействие необходимо).
6. Степень компетенции в области, к которой относится исправляемая ошибка.
7. Ваши собственные усилия по содействию сотрудникам, которые испытывают затруднения при исправлении ошибки.
8. Дополнительные усилия, выходящие за рамки рабочей необходимости.
9. Своевременность решения задач, не относящихся к текущему выпуску продукта.
0. Желание приобретать дополнительные сведения о продукте, преодолевая при этом трудности понимания и усваивая особо сложные моменты.
1. Настойчивость в решении задач, кажущихся невыполнимыми.

Естественно, отслеживать все эти моменты для каждой ошибки и каждого члена группы довольно сложно, однако рассматриваемый метод мониторинга — одного из фундаментальных принципов лидерства — конструктивен, а кроме того, придает деятельности по организации процесса развлекательный характер. Стоит также обратить внимание на то, как в вышеприведенном списке находит свое отражение концепция наставничества. Обычно, будучи лидером, вы стремитесь не допускать между разработчиками соперничества, способного усложнить достижение результата группой в целом. Именно поэтому при распределении вознаграждений так важно вести подсчет достигнутых сотрудниками успехов. Если бы я лучше разбирался в спорте, то, наверное, не удержался бы от какой-нибудь аналогии, например с количеством подстраховок (кажется, есть что-то такое в бейсболе или в баскетболе? или в обоих видах? не знаю).

Вне зависимости от того, какой метод поощрения сотрудников вы выберете, будьте справедливыми и не опаздывайте с вознаграждением. Когда какого-то богатого человека спросили о том, сколько денег ему нужно для счастья, он ответил: «Всегда немножко больше». Не забывайте, что вознаграждать нужно только тех, кто показал исключительные достижения, превысив все ожидания.

Не забывайте, что вознаграждать нужно только тех, кто показал исключительные достижения, превысив все ожидания.

Исправления

Необходимым условием профессионального роста любого программиста должна быть перепроверка результатов его деятельности. Исправляя ошибки, вы не должны действовать как университетский профессор и ставить жирный крест поверх неправильно решенной задачи. Настоящий лидер программистов старается сделать так, чтобы его подчиненные сами находили свои ошибки и учились на них. Конечно, при этом он должен советовать наиболее разумные способы решения проблем. Это очень тонкая задача, к решению которой лидер должен подходить со всей осторожностью, не впадая при этом в занудство. Недопустимо закрывать глаза на неаккуратный код и нерациональные действия сотрудников.

В главе 6, посвященной техническому лидерству, мы обсуждали метод критического обзора кода, а также последствия пренебрежения этой крайне важной обязанностью лидера. Вообще говоря, ваши корректирующие усилия должны быть в основном направлены на обеспечение максимальной отдачи всех сотрудников. Если кто-то работает одной левой, попытайтесь понять, почему. Не думаю, что ваш отдел так богат, что в нем можно держать сотрудника, который ограничивается исполнением своих минимальных обязанностей. Как я говорил в главе 3, иногда проблемы с сотрудниками решаются только путем увольнения. Если же проблема решаема, то доводить ее до необходимости увольнения неразумно. Значительно лучше попытаться регулярно оказывать сотруднику помощь, которая может привести к его реабилитации.

Пытаясь помочь программисту исправить ошибки или обрести мотивацию к активной деятельности, не забывайте, что процесс этот двунаправленный. Если вы не проверяете собственные ошибки, вряд ли к вашей правке будут относиться серьезно. Быть может, в человеческих отношениях действительно слишком много лицемерия, но лидеру программистов это качество противопоказано. В главе 2, говоря о борьбе с собственными слабостями, я упоминал о том, какое сильное влияние на сотрудников способен оказывать ваш стиль кодирования, — вопрос в том, какой характер носит это влияние: положительный или отрицательный? Ваши подчиненные прекрасно видят, насколько увлеченно вы относитесь к своим обязанностям, — укоряя кого-нибудь за недостаточное усердие, имейте это в виду.

Быть может, в человеческих отношениях действительно слишком много лицемерия, но лидеру программистов это качество противопоказано.

Предвидение

Способность предугадывать будущее развитие событий привлекает к вам последователей. Некоторые даже считают, что это — основное качество лидера. И действительно, оно играет очень важную роль — даже если ваших способностей провидца не хватает для создания очередной убойной программы. И самые скромные прогнозы чрезвычайно вдохновляют сотрудников. Представление о том, как сократить усилия по сопровождению в очередной версии флагмана вашей компании, может сыграть для нее не меньшую роль, чем изобретение Интернета.

Можно ли сформулировать метод предвидения? Быть может, настоящий провидец действительно общается с музами и сообщает людям услышанное? Наверное, при создании образцов высокой литературы и музыки без муз не обошлось, но вообще во всех творческих начинаниях очень важно мыслить нестандартно. Когда во время Второй мировой войны Алан Тьюринг (Alan Turing), работая на британскую разведку, расшифровывал немецкие сообщения, он попутно сформулировал новый способ осмысления вычислительных методов. В краткой биографии Тьюринга имеется следующий фрагмент:

> «Тьюринг изначально представлял себе, что его машина должна будет выполнять те же функции, что и человеческий мозг. [Тьюринг] предложил и проанализировал понятие «разумной машины»... Он обращался к аналогии с учителем и учени-

ком. Ученик может превзойти учителя, выйдя на более высокий интеллектуальный уровень, — для этого вполне достаточно информации, переданной ему учителем. По Тьюрингу, появление подобной машины вполне возможно — достаточно правил, введенных в машину. Впрочем, «играя против такой машины, чувствуешь, будто разум уничтожает что-то живое». Поскольку компьютер может учиться, его поведение выходит за рамки механистического детерминизма и демонстрирует свободу, создающую ощущение живого разума... Тьюринг мыслил на уровень выше математики, вычислимых чисел и даже компьютеров»[1].

К какому продолжению привели идеи, высказанные Тьюрингом, нам известно. Результатом стало признание исключительной важности алгоритмов в архитектуре программного обеспечения. Калькуляторы переросли в более мощные машины, способные побить чемпиона мира по шахматам. Благодаря нестандартности мышления Тьюринга его метод стал адекватным задачам современности. Роль лидера-прогнозиста как раз и предполагает такого рода широкое мышление; ваше положение в компании создает прекрасные условия для его воспитания.

Сочетание «глобального» осмысления деятельности вашей компании с техническими навыками и административной проницательностью позволяет выработать новые методы повышения продуктивности сотрудников. Такого рода практичное предвидение в нашей деятельности крайне приветствуется. Реализация навыков, которые вы почерпнули из этой главы, способна превратить прогнозы в реальность — передавайте свои знания, делегируйте полномочия, проверяйте ход выполнения проекта, участвуйте во взращивании и сборе урожая.

Реализация навыков, которые вы почерпнули из этой главы, способна превратить прогнозы в реальность — передавайте свои знания, делегируйте полномочия, проверяйте ход выполнения проекта, участвуйте во взращивании и сборе урожая.

Адаптация

В первой главе, как вы помните, я сделал акцент на адаптации к обязанностям лидера. Потребность в адаптации тем выше, чем дольше вы пребываете в роли лидера. Корректировка своих действий в соответствии с текущими условиями — это необходимость и одно из основных требований к хорошему лидеру. Люди, к сожалению, не могут предсказывать будущее в стратегической перспективе; будь у нас такая возможность, и планировать стало бы гораздо легче. В то же время люди как представители биологического вида привыкли адаптироваться к изменяющемся условиям среды. Не стоит ждать биологических изменений — лучше разделить свою лидерскую деятельность на ряд областей компетенции и оценить динамику развития своих навыков. Новые проблемы, встречающиеся в вашей работе, помогают тренировать способности к адаптации и приобретать новые знания. Любую трудную задачу следует рассматривать как возможность для дальнейшего развития, а отнюдь не как очередной повод «сжать кулаки». Эти мои размышления сильно напоминают концепцию позитивного мышления, о которой я мельком упоминал в предыдущей главе, но они разумны, а значит, заслуживают внимания.

[1] Paul Strathern, Turing and the Computer (New York: Doubleday, 1997), pp. 75–78.

Любую трудную задачу следует рассматривать как возможность для дальнейшего развития, а отнюдь не как очередной повод «сжать кулаки».

Пожалуй, я слишком увлекся разного рода клише. Среди практических вопросов адаптации, которыми вам стоит задаться, можно привести следующие.

- Каким образом нужно скорректировать мой стиль руководства, чтобы лучше решать текущие задачи?
- Какие новые технологии стоит изучить моим сотрудникам, чтобы наши продукты сохранили конкурентоспособность?
- Каких новых сотрудников мне стоит нанять, чтобы повысить компетентность персонала, а соответственно, и качество исполнения новых проектов?
- Можно ли изменить структуру отдела так, чтобы повысить качество, не срывая при этом сроков выхода продуктов на рынок?

Все эти вопросы (а может быть, и какие-то другие, которые придут вам в голову) нужно задавать себе (а желательно — и сотрудникам) ежедневно. Естественно, ответы на них должны быть выверенными, но не забывайте, что умение задавать нужные вопросы — это тоже элемент адаптации. Станьте экспертом по формулированию вопросов и поощряйте диалог с сотрудниками по поводу наиболее трудных текущих проблем.

Процесс адаптации идет строго по науке: сначала ставятся эксперименты, а затем их результаты оцениваются на предмет успешности. Неудачные методы заменяются новыми. Постепенно вырабатывается «идеальная» методология, которая в итоге переходит в ранг закона. Поощряя адаптационные процессы в самом себе и в своих сотрудниках, вы обрекаете себя на роль подопытных кроликов. Не стоит, впрочем, беспокоиться на этот счет — это не смертельно. При условии регулярной оценки результатов применения новых методов и их корректировки согласно текущей конъюнктуре вы сможете застраховать себя от фатальных последствий.

Пойдут ли за вами?

В чем заключается политическая опора вашей лидерской деятельности? Я не имею в виду выстраивание империй, но ведь должны быть у вас средства принуждения подчиненных к труду! Как я уже говорил, лидеры сами создают себе последователей, обращаясь при этом к своим наиболее притягательным качествам. В этом разделе мы поговорим о тех факторах общего характера, которые заставляют людей идти в заданном лидером направлении. Некоторые из представленных здесь методов очевидно выигрывают на фоне других, но, в принципе, все они имеют право на существование. Время от времени их приходится задействовать все сразу, и ничего страшного в этом нет. Попробуйте проанализировать, насколько они применимы в конкретных условиях, выберите для себя наиболее удачные — и вперед к победам!

Принуждение

Набирая в команду сотрудников, вы ставите перед ними ясное условие: чтобы получать зарплату, они должны трудиться. Это очевидное обстоятельство нередко воспринимается как данность. Действительно, современная система коммерческих отношений предполагает обмен мастерства на наличность. Но достаточно ли этого, чтобы заставить людей делать свою работу качественно? Как правило, нет, хотя с этого можно начинать — по крайней мере, в отсутствие более удачных средств мотивирования сотрудников.

Прибегать к силе принуждения нужно в последнюю очередь. Впрочем, рыночные условия иногда заставляют нас действовать подобным образом. Если компания сильно отстает в техническом отношении, для того чтобы отвоевать утерянные позиции, приходится предпринимать определенные усилия. Подобного рода внешние факторы, если донести их правильно, оказывают на людей должное воздействие. Не надо пугаться фраз типа «требования рынка» и «наши конкуренты располагают...» и соответствующих средств мотивирования — они работают!

Не надо пугаться фраз типа «требования рынка» и «наши конкуренты располагают...» и соответствующих средств мотивирования — они работают!

Не стоит прибегать к аргументам типа «потому что я так сказал» — если, конечно, вы не хотите, чтобы вас воспринимали как ментора с ограниченными умственными способностями. Не спорю, вы — начальник, и задания, которые вы распределяете между подчиненными, должны выполняться, но не забывайте — я пытаюсь научить вас пасти котов, а они, как известно, пугаются громких звуков.

Долг

Некоторые подчиненные следуют указаниям начальника просто потому, что не понимают, как можно этого не делать, — преданность у них в подкорке. Замечательно — вы должны поощрять такое отношение. С другой стороны, право называться боссом нужно заслужить, и одним лишь наличием должности здесь не обойтись. Повторюсь: высказывания о том, что сотрудники «обязаны следовать вашим указаниям», ни к чему не приводят. Это не более чем демонстрация грубой силы. Чувство долга формируется в человеке под воздействием личных и душевных факторов. Ему нельзя научить. Требовать его проявления в военных условиях допустимо, но в группе программистов подобное отношение практиковать не следует.

Как лучше всего воспитать чувство долга в своих сотрудниках? Нужно культивировать в них ощущение гордости за свои коллективные достижения. Все хотят быть причастными к победам. Взгляните на фанатов команды-победителя. Они ведь не принимают участия в состязании, но результат переполняет их эмоциями. Ваша задача в роли лидера — сделать так, чтобы сотрудники воспринимали долг перед командой, состоящей сплошь из выдающихся личностей, как высшую честь.

Как лучше всего воспитать чувство долга в своих сотрудниках? Нужно культивировать в них ощущение гордости за свои коллективные достижения. Все хотят быть причастными к победам.

Восхищение

Я восхищаюсь Альбертом Эйнштейном и поэтому, будь у меня такая возможность, пошел бы работать в любую лабораторию, если бы ей руководил он. Правда, свою лабораторию он держал в уме, так что это не самое удачное сравнение. Но вы меня поняли. Очень часто люди идут за лидером лишь потому, что восхищаются его личными качествами. Все это, конечно, замечательно, но вот незадача — большинство из нас не вызывают особого восхищения.

Способны ли вы работать так, чтобы подчиненные вами восхищались? Полагаю, да. Поможет ли это? Ну, с некоторой натяжкой тоже да. Правда, почитатели непостоянны — спросите у любого поклонника какой-нибудь закатившейся кинозвезды. Для того чтобы люди восхищались вами постоянно, да еще и получали от этого ощущения силы для своей деятельности, нужно не отступать от своего стиля руководства, неизменно демонстрировать эффективность лидерской деятельности и поддерживать стабильные отношения с персоналом; иными словами — проявлять последовательность. Это трудно, но необходимо. Последовательность — замечательное качество: она создает в подчиненных чувство защищенности и здорово помогает в самые трудные моменты, когда всей группе приходится биться над тем, чтобы закончить разработку проекта и сдать его в срок. Идти к такого рода последовательности нужно долго и упорно — не месяц и, вероятно, даже не год. Но, с другой стороны, ваши ежедневные усилия в достижении последовательности не пройдут незамеченными, а значит, вас будут больше уважать и стремиться работать прилежнее.

Для того чтобы люди восхищались вами постоянно да еще и получали от этого ощущения силы для своей деятельности, нужно не отступать от своего стиля руководства, неизменно демонстрировать эффективность лидерской деятельности и поддерживать стабильные отношения с персоналом; иными словами — проявлять последовательность.

Вознаграждение

Многие считают, что вознаграждение — это один из лучших способов принуждения. В разделе, посвященном фундаментальным принципам лидерства, я уже говорил о премиях. Не стоит забывать, что вознаграждение — это краткосрочное решение, оно может мотивировать только на первых порах. Увлекаясь раздачей наград, мы начинаем регулярно задаваться неблагодарным вопросом: «А что ты для себя сегодня сделал?» Лично я терпеть не могу так мыслить. Итак, ценность тех видов вознаграждения, к которым руководители прибегают чаще всего (деньги, выходные и т. п.), ограничена во времени.

Нужно сделать так, чтобы сотрудники извлекали из работы над проектами, которыми вы руководите, особые, неосязаемые виды вознаграждения — ведь они самые эффективные! Бытует мнение о том, что программирование как область

человеческой деятельности дарит своим приверженцам наслаждение — но этого мало. За счет своих навыков руководителя и лидера вы должны стремиться создать в своем отделе такую атмосферу, в которой сотрудники, просыпаясь утром, могли бы благодарить судьбу за то, какая у них прекрасная работа. Не слишком ли высокую планку я перед вами ставлю? Ну, возможно, для некоторых она и окажется недостижимой, но, в принципе, стремиться к этой цели стоит — результаты оправдывают все усилия.

Как этого достичь? Ответ вам известен, но я повторюсь, ибо повторение, как известно, — мать учения: *концентрируйтесь и становитесь лидером*. О том, как концентрироваться, мы уже говорили в предыдущих главах — с тем, что касается принципов лидерства, мы бьемся уже здесь. Испытывайте на практике теоретический материал, который я здесь излагаю, и ищите наиболее подходящие в вашей ситуации методы. Воистину практика — это путь к совершенству, и вашим сотрудникам она приносит серьезные дивиденды.

Знание

Про нашу индустрию можно с уверенностью сказать: «Знание — сила!» Программисты любят работать под началом компетентных лидеров, поскольку те помогают им решать их собственные задачи. Действительно, знание — это хорошее средство создать последователей. Станьте экспертом в своей области, и программисты, готовые идти по заданному вами пути, не заставят себя ждать. Не слишком ли многого я требую от вас, учитывая необходимость ежедневно выполнять множество административных функций? Да, это трудно, но если вы остановитесь в своем профессиональном развитии, то не сможете должным образом руководить своими подчиненными.

В предыдущей главе я упоминал о программистах, которые, обладая блестящей квалификацией, тем не менее не приспособлены к выполнению организационных задач. Если вы — гений (и при этом чувствуете себя на своем месте), мои вам поздравления. В таком случае у вас, скорее всего, уже есть последователи. Но ведь их еще нужно удержать — для этого старайтесь передавать им свои знания, которые позволили бы им дорасти до вашего уровня. Я это серьезно. В нашей индустрии много блестящих программистов и лидеров, но нужно еще больше. Обращайте особое внимание на свои наставнические функции, и вам удастся воспитать достойных последователей.

Возрастные аспекты лидерства

Хочется надеяться, что эту книгу читают люди самых разных возрастов. Сам я пребываю в среднем возрасте; вы же, быть может, лишь недавно вышли из юношеского состояния. Биологический возраст оказывает довольно серьезное воздействие на осмысление лидерской роли и подход к решению задач в области разработки программных средств. Не так уж часто встречаются опытные лидеры, не успевшие еще разменять четвертый десяток, хотя есть и такие. Старость — отнюдь не гарантия мудрости. У нас, «стариков», часто встречаются не слишком приятные привычки, от которых нужно избавляться, и в этом нам совсем не помешают рекомендации молодых людей.

Какие черты характерны для молодых?

- Они быстро усваивают новую информацию.
- Они на равных с новыми технологиями.
- Для них нет невыполнимых задач.
- Они ощущают себя бессмертными, что позволяет им добиваться великих достижений.

Впрочем, кто сказал, что все эти качества не могут относиться к более взрослым людям? Взрослея, вы должны обращать внимание на поведение молодых, поскольку нам есть чему у них поучиться.

Теперь перечислю качества, которыми обычно наделяют людей в возрасте (я не имею в виду какой-то конкретный возраст).

- Они мудры, потому что прожили долго и всю жизнь учились.
- У них есть многолетний опыт общения с изменяющейся технологией, а следовательно, они умеют к ней адаптироваться.
- Они более терпеливы, поскольку научены на собственных ошибках.
- Они все больше ощущают свою смертность и пытаются выжать максимум из каждого прожитого дня.

Опять же нельзя утверждать, что молодые люди лишены этих качеств. Мы все можем учиться на чужом опыте. Будь вы самым молодым или, наоборот, самым старшим из всей команды — в любом случае у окружающих есть чему поучиться. Способность к обучению не имеет возрастных рамок. Отказ от повышения уровня знаний в нашей отрасли чреват закатом карьеры.

Мы все можем учиться на чужом опыте. Будь вы самым молодым или, наоборот, самым старшим из всей команды — в любом случае у окружающих есть чему поучиться. Способность к обучению не имеет возрастных рамок. Отказ от повышения уровня знаний в нашей отрасли чреват закатом карьеры.

Есть и другие, более серьезные возрастные различия. В современной психологии, которая являет собой науку о поведении, проведено несколько исследований жизненного цикла человека. Согласно этим исследованиям, возраст действительно обусловливает определенный подход к рабочей деятельности, и в особенности это касается лидерства. Следующий пассаж, в котором жизнь, подобно солнечному году, трактуется как последовательность сезонов, доказывает различия в ментальности в зависимости от возраста.

«Если проводить аналогии с временами года, нетрудно заметить, что жизнь обладает определенными очертаниями и проходит через ряд четких форм. Сезон — это относительно стабильная часть общего цикла. Лето существенно отличается от зимы, равно как и сумерки не похожи на зарю. Утверждение об относительной стабильности сезона, впрочем, еще не означает, что он неизменен или статичен. Каждому сезону — свое время; каждый из них важен не меньше, чем другие, и требует осмысления с привлечением индивидуального понятийного аппарата. Нет сезона, который можно было бы признать менее значимым, чем любой другой. Каждый из них занимает особое и необходимое

положение в рамках целого и придает ему собственный оттенок. Любой сезон представляется органичным элементом общего цикла — соединяя прошлое с настоящим, он одновременно заключает в себе то и другое»[1].

В зависимости от того, какой сезон мы переживаем в данный момент, наши воззрения на лидерство меняются. Вместе с ними меняется восприятие важности успехов и неудач. Кроме вас, никто не может сказать, каким конкретно образом возраст влияет на ваше представление о лидерской роли. Не существует такой закономерности, на основании которой мы могли бы утверждать, что один возраст лучше другого.

Я тоже не избежал влияния возрастных факторов. Свой путь в индустрии я начал с кодирования на Фортране, используя перфокарты для IBM; я передавал колоды карт через конторку в своем компьютерном центре, а чтобы узнать, работает ли программа, приходилось ждать несколько дней. Мне довелось работать с первой предтечей Интернета — сетью, которая в те дни называлась ARPANET[2], — она работала жутко медленно и понимала только символы. Поэтому я причисляю себя к первому поколению «ботаников» — тех, кто в колледже не мог обходиться без логарифмической линейки за поясом и запасных предохранителей в кармане. Сегодняшнего лидера программистов порой невозможно отличить от какого-нибудь магистра. Он сам зачастую носит гордое имя магистра экономики и управления (MBA) и, конечно, имеет за плечами серьезное образование в области компьютерных наук. Представителям моего поколения приходилось получать все эти знания на ходу. Подобного рода различия, разумеется, оказывают влияние на наши взгляды и на решения, которые мы принимаем в роли лидеров. С другой стороны, к какому бы поколению мы ни принадлежали, мы можем учиться на своих различиях. Именно в этом взаимодействии рождаются самые удачные воззрения и идеи.

Как лидеру сочетать форму и содержание

Форма — это выраженные в поведении личностные характеристики, которые в нашем случае переплетаются с реализацией принципов лидерства. Форму, которая безусловно важна, нужно удачно сочетать с содержанием, которое отражает ваши представления о важнейших лидерских принципах. Форма без содержания — это не более чем голая методика, лишенная стратегической ценности. Тем не менее мы зачастую раньше замечаем в окружающих форму, чем начинаем разбираться в ее наполнении. Слава богу, не существуют двух лидеров, имеющих одну и ту же форму. Многообразие для технарей чрезвычайно полезно, и будучи лидерами, мы не менее разнообразны, чем программисты, которыми руководим.

Все мы учимся походить на лидеров, которым следуем и которыми восхищаемся. Хорошо бы еще учиться на собственных достижениях и ошибках в роли лидеров. Литература по коммерческому менеджменту изобилует биографиями людей,

---

[1] Daniel J. Levinson, The Seasons of a Man's Life (New York: Ballantine Books, 1978), p. 7.

[2] ARPA — это сокращение от Advanced Research Projects Agency (Управление перспективных исследовательских программ) — ведомства, которое в 1970-х годах было переименовано в DARPA. Первая буквы D в этой новой аббревиатуре обозначала принадлежность к министерству обороны США (defense — оборона), заведовавшего ракетами, бомбами с лазерным наведением, технологией «Стелс» и всеми прочими прелестями времен холодной войны.

примерами которых можно восхищаться. В главе 2 я уже ссылался на Джека Уэлча (Jack Welch) — бывшего главу General Electric и одного из тех людей, которым не стыдно подражать. Есть в нашей индустрии еще два хрестоматийных лидера: Энди Гроув (Andy Grove) и Билл Гейтс (Bill Gates).

Энди Гроув — агрессивный параноик

Под руководством Гроува компания Intel превратилась в крупнейшего в мире производителя полупроводников и одну из самых уважаемых организаций. Гроуву удалось привести Intel к такому статусу вопреки постоянным переменам и жесткой конкуренции, наблюдавшимися в нашей индустрии на протяжении последних десятилетий. Как это у него получилось? В первой главе своей книги о лидерстве Гроув пишет:

> «Мне часто приписывают крылатую фразу "выживают только параноики". Хоть убейте, не вспомню, когда я ее первый раз произнес, но, действительно, в том, что касается бизнеса, я верю в силу паранойи. Успех коммерческого предприятия одновременно создает предпосылки для его разрушения. Чем успешнее ты становишься, тем больше людей пытаются урвать кусочек твоего дела, потом другой — и в конечном итоге оставить тебя без гроша в кармане. Я убежден, что руководитель должен в первую очередь постоянно отбиваться от нападок завистников и прививать аналогичные настроения своим подчиненным»[1].

Паранойя — в том качестве, в котором ее определяет Гроув, — действительно демонстрирует в бизнесе чрезвычайную эффективность. Вести себя подобным образом с женой, конечно, не стоит, но вот в Intel под руководством Гроува эта стратегия сработала. Познакомившись с тем, как Гроув управлял Intel, вы обязательно придете к выводу, что основным фактором успеха компании стал стиль реагирования на перемены, практикуемый ее руководителем. Управлять переменами очень сложно, а в роли лидера вам предстоит сталкиваться с этой проблемой ежедневно.

Гроув умел предвидеть перемены и атаковал их. Даже несмотря на неудачи (вспомните, например, историю с плавающей точкой в процессорах Pentium), он осознавал основополагающие цели своей индустрии и творчески реагировал на самые неожиданные явления. Итак, учитесь: не ставьте себя в изолированное положение — постоянно следите за факторами, которые оказывают влияние на индустрию в целом и на ваших сотрудников в частности. Если для этого придется стать параноиком — станьте им! Лучше ожидать неожиданное, чем удивляться, когда оно предстанет перед вами в полный рост.

Не ставьте себя в изолированное положение — постоянно следите за факторами, которые оказывают влияние на индустрию в целом и на ваших сотрудников в частности. Если для этого придется стать параноиком — станьте им! Лучше ожидать неожиданное, чем удивляться, когда оно предстанет перед вами в полный рост.

Билл Гейтс — одержимость и расчетливость

Трудно не восхищаться Биллом Гейтсом. Сравните выгнанного из колледжа юношу с неадекватным выражением лица, запечатленного на фотографии 1978 года,

[1] Andrew S. Grove, Only the Paranoid Survive (New York: Random House, 1996), p. 3.

с тем человечищем, которым он стал через несколько лет, и восторга не избежать. Корни империи Microsoft — в одержимости, предвидении, предпринимательских способностях и расчетливости одного из ее основателей. О нем ходят легенды. Один из соседей Гейтса по общежитию в колледже рассказывает, как тот играл в покер, буквально следующее:

> «У Билла было одно совершенно маньяческое качество... За все, что его интересовало, он брался мертвой хваткой и не отрывался. У него была цель — разобраться в предмете любой ценой. Наверное, глупо сравнивать покер и Microsoft, но в обоих случаях Билл направлял всю свою энергию на решение одной задачи, и ему было все равно, что об этом думают окружающие»[1].

Остальное — история. Впрочем, если вы не понимаете, что одержимость успехом в конечном итоге приводит к успеху, значит, вы недооцениваете один из важнейших аспектов лидерства. Как вы уже знаете, важнейшим принципом деятельности лидера я считаю умение концентрироваться. Этим искусством в совершенстве владеют одержимые. Возможно, мотивация лидера вызывает вопросы, но успеху, от которого выигрывают очень многие, трудно что-то противопоставить.

К стилю лидерства, который практикует Гейтс, я также отношу расчетливость. Как и Гроув, он умеет предсказывать перемены и к моменту их наступления уже имеет четкий план действий. Вот, например, вы замечали, что в большинстве версий Windows период действия авторских прав указывается с 1981 года[2]? Это, по-моему, прекрасное свидетельство планирования, которое этот чрезвычайно расчетливый человек совершенствовал в течение многих лет. Да, Гейтс не всегда лидировал в индустрии, но в любом случае на нашу повседневную деятельность в роли программистов и их лидеров он оказал наибольшее влияние. Вот где формальная расчетливость в сочетании с содержательной стороной дела приводит к формированию лидерского ресурса. Любые инструменты и инфраструктура, созданные вне Microsoft, все равно оцениваются в свете достижений этой компании (или вопреки им).

Вы — _____ (введите недостающее слово)

Исходя из двух вышеприведенных примеров лидерства, как бы вы охарактеризовали свой лидерский стиль? Впрочем, лучше мотайте на ус следующую мысль: будучи лидером, вы можете подбирать подчиненных, исходя из своих знаний и интуитивной оценки кандидатов. А что если перевернуть ситуацию вверх дном и предоставить вашим подчиненным возможность выбирать тип лидера? Предположим, что сотрудники могли бы интервьюировать нескольких лидеров и брать на работу того, который им понравится. Ситуация, конечно, довольно сложная и странная, но задать себе такой вопрос необходимо — он имеет непосредственное отношение к анализу лидером своей деятельности (см. главу 2) и отрицательным чертам порочных стилей лидерства, которые мы обсуждали в предыдущей главе (см. главу 7).

Только вы и никто иной можете заполнить оставленное в заголовке пустое пространство. Вводимый текст зависит от вашего представления о собственном стиле

---

[1] See James Wallace and Jim Erickson, Hard Drive: Bill Gates and the Making of the Microsoft Empire (New York: John Wiley & Sons, 1992).

[2] Ibid., p. 61.

лидерства. Я также не могу говорить от лица ваших подчиненных и утверждать, что из нескольких кандидатур они выбрали бы именно вас. Это вопросы личного характера, на которые вы должны отвечать только сами. Впрочем, приведу одну цитату, которая в данном контексте, полагаю, весьма полезна:

> «Качества лидера нельзя почерпнуть из выступлений на семинаре или с полки вашего любимого книжного магазина; лидерство неизмеримо в долларах и центах. Лидерство происходит из деятельности в интересах команды — деятельности разумной и направленной на достижение стратегических коммерческих целей. Поймите, что люди — это, по существу, средства производства, и ваш успех зависит от уважения к вам со стороны подчиненных, которых вы призваны защищать от происков внешних сил»[1].

Предвижу иронию в связи с этой цитатой — ведь и мою книгу вы, наверное, купили в местном магазине. Если это приобретение помогло вам осознать смысл сочетания формы и содержания, значит, оно не было напрасным. «Происки» упоминаются здесь не совсем в том смысле, в котором о них рассуждал Макиавелли; тем не менее в духе здоровой паранойи вы должны готовиться к грядущим переменам и не позволять им застать вас врасплох.

Резюме

Не удивляет ли вас, что в книге, в которой всего десять глав, я подобрался к сути лидерства лишь к концу восьмой? Дело в том, что продавая вам свое представление о лидерстве, я совершаю своего рода пакетную сделку. Чтобы добраться до подарка, нужно снять несколько слоев обертки. Кроме того, если уж мы заговорили об обертках, замечу, что я, не отставая от других авторов, попытался изложить материал четко и понятно. Известно, что ясность изложения — один из определяющих факторов успеха книги. Впрочем, проницательный ум всегда сможет отличить форму от содержания. Вот почему многие считают, что телевизионные программы, основанные на реальных событиях, выигрывают в сравнении с традиционными драматическими формами. По-моему, такие программы — это не совсем удачная попытка показать реальную жизнь, которая страдает из-за своей замкнутости. Как говорится, недокументированная жизнь не стоит того, чтобы ее проживать. (Это, типа, шутка.)

Лидерство формируется в практической деятельности

Основная мысль, которую я хотел донести в этой главе, сводится к тому, что путь к лидерству индивидуален для каждого человека и зависит от конкретных условий: компании, в которой вы работаете, сотрудников, которыми руководите, времени, в котором живете. Без теории не обойтись, но лишь практика вдыхает в нее жизнь.

---

[1] Don S. Olson and Carol L. Stimmel, The Manager Pool: Patterns for Radical Leadership (New York: Addison-Wesley, 2002), p. 9.

Через несколько лет вы, возможно, напишете собственную книгу или статью о лидерстве — это полезно хотя бы потому, что дает возможность упорядочить свой опыт. И в этом контексте очень важно обращаться к практике документирования. Если вы положительно относитесь к этому занятию, заведите дневник и фиксируйте в нем свою деятельность. Если вам понравилась моя книга, можете рассказать о ней знакомым. Тогда она станет бестселлером, и у меня появится возможность больше публиковаться[1]. Я мог бы собрать присланные вами материалы в сборник статей о лидерстве. Не подумайте, что я пекусь лишь о собственных интересах, — я действую в соответствии с изложенной выше стратегией наставничества. В нынешней поре моей жизни она прочно вошла в мои лидерские будни.

Отталкивайтесь от основных принципов лидерства

Переходя к реальной практике лидерства, не забывайте фундаментальные принципы, которыми я с вами поделился. Ими пользуются многие мои коллеги и учителя. Не жадничайте и делитесь с окружающими знаниями, которые, по вашему мнению, правдивы. Рассказывая правдивые истории, вы сможете зафиксировать накопленный опыт и утвердить его в своем характере.

Во всем, что вы делаете, необходимо соблюдать пять основных принципов лидерства.

- *Понимание*. Определитесь с тем, куда вы идете.
- *Передача знаний*. Делитесь своими знаниями — так, чтобы подчиненные все поняли.
- *Делегирование*. Общие задачи нужно решать общими усилиями.
- *Проверка*. Проверяйте свои действия и то, что делают ваши подчиненные в контексте достижения поставленных целей.
- *Участие*. Погружайтесь в работу с головой — будьте примером для остальных.

Не забывайте о надстройке. Свое развитие основные принципы лидерства получают в следующих областях деятельности.

- *Наставничество*. Учите окружающих учить.
- *Вознаграждение*. Награждая сотрудников за выдающиеся результаты, вы сможете создать ситуацию, в которой одни успехи естественным образом выливаются в другие.
- *Исправление*. Помогая сотрудникам учиться на собственных ошибках, повышайте их квалификацию.
- *Предвидение*. Предугадывайте проблемы, пока они не ударили по вашей команде, — это станет хорошим стимулом для окружающих.
- *Адаптация*. Совершенствуйтесь, извлекая уроки из собственных ошибок.

---

[1] Я серьезно! Присылайте рассказы о своем пути на лидерском поприще по адресу HerdingCats@mindspring.com.

Что дальше

В следующей главе мы поговорим об отношениях с начальством. Попутно вы сможете оценить свои лидерские способности. Одно дело вести подчиненных в заданном направлении, и совсем другое — самому следовать за боссом. Чтобы изучить проблему лидерства в комплексе, нужно иметь опыт следования за лидером. Знакомясь с материалом следующей главы, не забывайте принципы, изложенные в главе текущей, и оценивайте, в какой степени их воплощает ваш начальник. Вероятно, из этих наблюдений вы сможете почерпнуть много полезного — ведь, скорее всего, ваш шеф уже прошел через все трудности и радости, о которых я вещал в этой главе, и неоднократно имел возможность пересмотреть и усовершенствовать свои лидерские навыки.

Глава 9

Как ужиться с начальством

В этой главе, чтобы выдержать четкую повествовательную линию, я буду говорить не о начальниках, а о начальницах. Есть у меня и другое соображение. Традиционно в нашей индустрии доминируют мужчины, а женщины рассматриваются в процессе разработки программных средств несколько в другом ракурсе. Возможно, если бы в период формирования нашей области знаний женщины были в ней представлены более широко, нам удалось бы избежать некоторых очевидных огрехов, которые время от времени мы допускаем.

Не пугайтесь названия этой главы, даже если ваши отношения с вышестоящей инстанцией строятся не слишком удачно. Осознав, какое давление испытывает начальница, вы сможете сделать свои отношения с ней более конструктивными, извлечь из них пользу для самосовершенствования в роли лидера и развития способностей собственных подчиненных. Не забывайте, что у нее, по-видимому, тоже есть начальник и она не раз ощущала себя в вашей шкуре.

Если отношения с начальницей у вас стабильно ровные, надеюсь, что идеи, которые я намерен изложить в этой главе, окажут на вас сильное эмоциональное влияние. Вы с вашей начальницей должны составить прекрасный альянс. Прочность и гибкость рабочих отношений с начальницей — это один из тех факторов, которые способны ускорить движение к лидерству. Лидер тоже должен уметь следовать в четко заданном направлении; эти навыки помогут вам выстроить план действий относительно собственных подчиненных в контексте решения установленных вами и вашей начальницей приоритетных задач.

Как понять, чем живет ваша начальница

Как легче всего сбалансировать отношения с начальницей? Все очень просто: представьте, что вы сами продвинулись вверх по карьерной лестнице. Возможно, вы совершенно не собираетесь этого делать — в конце концов, если вы пришли в руководящее звено из числа программистов, как это обычно и случается, вряд ли вы

испытываете неудовлетворенность своим положением. И тем не менее вообразите себе, что вы продвинулись до должности, в данный момент занимаемой вашей нынешней начальницей. Попробуйте представить ожидания, которые в этом случае вы предъявляли бы своим подчиненным. Если уж вы вынуждены выстраивать совместную деятельность вместе с начальницей, этот и подобные ему вопросы задавать себе совершенно необходимо.

Поймите меня правильно — воображать, как бы вы командовали своей начальницей, не стоит; просто представьте, что она управляет вами точно так же внимательно, как вы управляете сотрудниками своего отдела. Желание порадовать начальницу естественно. Впрочем, будьте осторожны: если не отдавать себе полный отчет в своих мотивах, есть опасность в порыве служебного рвения забыть о необходимости быть честным. Что заставляет начальников радоваться? Выполнение выданных ими заданий в срок, без жалоб и нытья. Именно результаты должны стать мерой оценки вашей деятельности, а отнюдь не успокаивающие речи, направленные лишь на то, чтобы просто слегка отсрочить приговор.

Принципы, которые я здесь излагаю, допускают незначительную корректировку в зависимости от структуры конкретной компании. Впрочем, вы должны усвоить основную мысль: чем выше положение в управленческой иерархии, тем больше ответственность и тем шире область деятельности, на которую она распространяется. Вместе с ответственностью повышается потребность в делегировании, которое в таком случае проходит уже несколько уровней бюрократии. Действительно, цепочка делегирования, в которой вы исполняете роль одного из звеньев, крайне трудно поддается организации и проверке.

Чем выше положение в управленческой иерархии, тем больше ответственность и тем шире область деятельности, на которую она распространяется.

Авторы книги «The Centerless Corporation»[1] высказывают следующее наблюдение: «В то время как размеры современных корпораций стремительно растут, их управляемость в плане достижения заданных целей неуклонно падает». Имейте в виду: ваша начальница пытается бороться с этой тенденцией. «Заданными целями» должны проникнуться все сотрудники компании. Чем вы можете помочь, чтобы достичь такого результата?

Несмотря на все разговоры о «децентрализованных» корпорациях и необходимости модернизации структуры корпораций, большинство из нас продолжает работать в рамках иерархических структур, в которых один отчитывается перед другим, другой — перед третьим и т. д. Субординация — это отнюдь не пустое слово. Сколько отчетов вынуждена подавать ваша начальница, а сколько — вы? Попробуйте посчитать. Это — лишь один из тех факторов, которые формируют представление о роли начальницы в вашей рабочей деятельности. Вы ведь прекрасно знаете по собственному опыту, что управлять людьми не так уж просто; то обстоятельство, что у нее значительно больше подчиненных, уже о чем-то говорит.

[1] Bruce A. Pasternack and Albert J. Viscio, The Centerless Corporation (New York: Simon & Shuster, 1998), p. 15.

И здесь математические расчеты не всегда приводят к верным выводам. Руководители, которые стоят в иерархии выше вас, ответственны за *всех* нижестоящих. При общении с начальницей вы должны иметь это в виду. Ее уровень ответственности значительно выше вашего.

Честность и принципиальность против подтасовок и лжи

В предыдущем разделе я мельком упоминал о честности — не хотелось бы продолжать эту тему, но придется. При любой попытке утверждения реалистичных сроков сдачи проекта честность вступает в противоборство с прожектерством. Большинство компаний бредят маркетингом; срок выхода на рынок — их основной приоритет. Для того чтобы сохранить положение на рынке, необходимо в процессе усовершенствования программных продуктов исходить именно из бизнес-требований. Такая ситуация накладывает на ваших сотрудников дополнительные обязательства, которые, естественно, передаются и вам. Напряжение, испытываемое вашей начальницей, еще серьезнее — ведь, скорее всего, она тоже кому-то подотчетна. Эта цепочка подчинения — очень серьезный фактор; и будьте уверены, утверждения о важности требований рынка совсем не преувеличены. Взять хотя бы мощнейшую маркетинговую машину Microsoft — эту компанию можно не любить, но с ее успехом не поспоришь[1]. Спросите директора любой компании или группу заинтересованных лиц: хотят ли они такого же успеха, как Microsoft? Вряд ли ответ будет отрицательным.

Находясь под влиянием рыночной конъюнктуры, компании нередко устанавливают сроки сдачи продуктов, не советуясь с руководителями вашего уровня. Ситуации, когда утверждение бизнес-плана предшествует окончательной формулировке коммерческих требований, случаются сплошь и рядом. В главе 3 я уже говорил о различиях между реалистичными и нереалистичными планами проектов. Полагаю, соответствующие принципы следует воспроизвести и здесь. В идеале планирование должно осуществляться в такой последовательности:

1. Утверждение коммерческих требований.
2. Создание проектного решения, допускающего успешную реализацию в продукте всех требований.
3. Макетирование проектного решения с целью выявления его недостатков и последующей корректировки проектного решения или требований.
4. Планирование проекта с учетом сроков разработки и тестирования.

Разработанный план позволяет с определенной уверенностью говорить о временных рамках выпуска; исключительно на их основе можно давать какие-либо обещания представителям отдела продаж. Естественно, как и во всех проектах, временные рамки выпуска должны быть обусловлены успешным бета-тестированием.

---

[1] Ну то есть поспорить можно, но какой в этом смысл? Я много лет пользуюсь продуктами Microsoft и не испытываю особых затруднений — полагаю, у вас та же ситуация.

Как известно, мир, в котором мы живем, несовершенен; иначе вряд ли было бы столько разговоров о программистах, у которых под столами спальные мешки. У вас, таким образом, есть единственный выход — научиться выживать в реальных условиях[1]. При чем тут честность, спросите вы? При том, что вы должны осознавать нереалистичность поставленных перед вами задач в свете реальных условий, примеры которых перечислены ниже:

- несмотря на то что коммерческие требования сформулированы еще не полностью, в погоне за соблюдением неизвестно кем установленной даты выпуска вы вынуждены приступать к проектированию немедленно;
- из-за неразберихи с требованиями вам приходится постоянно корректировать проектное решение;
- у вас не остается времени на макетирование, или, что еще хуже, недоработанный макет превращается в код;
- единственный план, которым вы располагаете, заключается в том, чтобы, отталкиваясь от установленной специалистами по продажам даты выпуска, пытаться получить представление о реальных сроках разработки.

В любом случае вопреки объективной реальности вы должны всеми силами стремиться к тому, чтобы закончить работу в срок. Юность нашей индустрии и давление рыночных факторов заставляет нас совершать героические поступки. Таким образом, честностью я называю способность признаться самому себе в трудности задачи, но все-таки попытаться ее решить. Вы со мной согласны? Если согласны, присылайте резюме на мой адрес — такие, как вы, мне необходимы.

Еще пара слов насчет геройства[2]. В начале карьеры идея стать героем воодушевляет, однако реализуют ее немногие. На самом деле стремиться нужно к балансу между ожиданиями и реальными действиями, исходя при этом из соображений честности. Иначе говоря, если вы пару раз сорвете сроки, никто не удивится. Это исправимые вещи. Значительно труднее справиться с привычкой давать невыполнимые обещания. Если вы уж обещаете что-то, не забывайте в полной мере доносить до сведения начальства все те факторы, которые могут воспрепятствовать реализации плана. Любые обязательства должны быть подтверждены в проекте с некоторой долей уверенности, которая варьируется в зависимости от ситуации. Восстановить репутацию значительно сложнее, чем исправить ошибки в выпущенном продукте.

На самом деле стремиться нужно к балансу между ожиданиями и реальными действиями, исходя при этом из соображений честности. Восстановить репутацию значительно сложнее, чем исправить ошибки в выпущенном продукте.

Обратной стороной геройства является фатализм. Фаталист — это человек, который неудачно пытался стать героем, а потом в ситуации, которую он создал

[1] Рекомендую в этом контексте ознакомиться с предложениями Йордона (Yourdon) по вытягиванию проектов, изложенными в книге «Death march» (см. библиографию).

[2] Лишь те, кто живет в обществе, подобном нашему, могут претендовать на звание героев. Впрочем,считайте, что вам повезло получить хорошую работу, и старайтесь выкладываться на все сто. Настоящие герои умерли одиннадцатого сентября в попытках спасти невинных жертв.

сам, начинал винить судьбу. Если вам приходится иметь дело с нереальным проектом, необходимо отдавать себе отчет в том, что фактически вы и ваша команда попадаете в условия, приближенные к боевым. Регулярная работа по ночам изматывает разработчиков, в результате страдает код. Не стоит вводить в проект новых программистов — если это случится на поздней стадии разработки, вы рискуете в очередной раз подтвердить закон Брукса[1]. Именно по этой причине на начальных этапах планирования проекта так важна честность (если, конечно, у вас есть план).

Честность зависит от тщеславия и чувства собственного достоинства. Я уже в том возрасте, когда каждое утро для удовлетворения тщеславия мне нужны лак для волос, зеркало и время. Признаться, я лысею, и это — горькая правда. Я могу пытаться это скрывать, но ведь все равно окружающие узнают![2] Аналогичным образом лучше высказать голую правду о сроках, чем сочинять сказки.

Как помочь начальнице удачно спланировать процесс

Начальница знает, что и когда нужно делать; вы специализируетесь на том, как этого достичь. Так... кажется, я выразился не слишком ясно. Попробуем еще раз. Как правило, планирование в масштабах предприятия проводится начальством; вы же призваны заменить общие наметки детальным планом. Час от часу не легче, так? То-то же. Планирование подчиняется формуле: два шага вперед, один шаг назад. Процесс этот напоминает рекурсивную процедуру: нужно постоянно «копать» стратегический план, наполняя его деталями, которые, кстати, вам же и предстоит реализовывать. В этом отношении вы можете оказать начальнице ценную услугу — высказать свое экспертное мнение по поводу ее глобальных задумок. В планировании кропотливая работа оттесняет вдохновенные прозрения.

В некоторых компаниях отдел разработки рассматривается как производственный цех, который, принимая на входе спецификации, в конечном счете выпускает готовый продукт. Будь это правдой, я, наверное, предпочел бы сменить профессию и из наемного рабочего превратиться во владельца такой фабрики. Во многих консалтинговых компаниях на полном серьезе рассуждают о «фабриках программных продуктов». Большинство таких компаний базируются за океаном, а выводы свои строят исходя из высокой стоимости и низкой окупаемости инвестиций — действительно, многие отделы разработки программного обеспечения в американских компаниях демонстрируют такую динамику. В двух книгах Эдварда Йордона (Edward Yourdon) — «Decline & Fall of the American Programmer» (Yourdon Press, 1993) и «Rise & Resurrection of the American Programmer» (Yourdon Press, 1996) — раскрываются причины, по которым в кругах разработчиков программного обеспечения планирование зачастую проводится недостаточно тщательно или вообще игнорируется. Возможно, связано это с тем, что мы упорствуем

---

[1] «Введение в запаздывающий проект новых сотрудников увеличивает время разработки» — см. Brooks, op. cit., p. 25.

[2] Если вы женщина, стали бы вы встречаться с типом, который, стараясь скрыть правду, обматывает голову последним оставшимся волосом, как тюрбаном? Не думаю. Хотя если вы знаете даму, которая на это пойдет, дайте телефончик.

в восприятии программирования как одного из видов искусства[1]. Если бы мы подготавливали запуск ракеты на Луну, уверяю вас — обойтись без планирования было бы невозможно.

Кстати, скажу несколько слов об американской космонавтике. Как могло произойти, что в 1960-е годы, когда существующие программные средства уступали уровню типичного современного карманного компьютера, нам удалось высадить человека на Луну? Секрет в том, что причастные к этому проекту специалисты, исходя из имеющихся инструментальных средств, планировали свою деятельность с расчетом на успешный финал. В своих мемуарах, рассказывая о Центре управления полетами, Джин Кранц[2] (Gene Kranz) раскрывает принципы, которые, по его мнению, определили заметные успехи подведомственной ему структуры.

- *Дисциплина*. Способность лидировать, с одной стороны, и идти в заданном направлении, с другой. Понимание того, что для решения задачи необходимо, прежде всего, совладать с собой.
- *Компетентность*. Космические проекты не терпят небрежности и безразличия — требуется полная готовность к выполнению задания и тотальная устремленность на успех.
- *Уверенность*. Вера в себя и окружающих; сознание необходимости задушить страх и неуверенность.
- *Ответственность*. Понимание того, что поставленную задачу нельзя никому передоверить; есть всего две альтернативы: либо сделать то, что требуется, либо потерпеть фиаско.
- *Упорство*. Нацеленность на преодоление возможных трудностей; последовательность в достижении цели, даже если для этого необходимо пройти по сложному пути.
- *Командные усилия*. Уважение к способностям друг друга и их разумная эксплуатация; ощущение работы над общей целью и коллективной ответственности за результат.

Далее Кранц утверждает, что «лучше попробовать и потерпеть неудачу, чем приложить недостаточно усилий». Придерживаясь этих принципов, он разрабатывал прекрасные планы — иногда слишком поспешно, но тем не менее ему это удавалось. Он не мог действовать без планирования — в конце концов, на него ложилась ответственность за человеческие жизни. Программные продукты, конечно, никого не убивают, но неудачный результат разработки способен сломать вашу карьеру вместе с карьерой начальницы[3].

---

[1] Каюсь, в этой книге я слишком вольно обращался с терминами «мастерство» и «искусство» и недостаточно акцентировал ваше внимание на том, что разработка программных средств — это, как-никак, инженерная дисциплина. Но такой выбор был осознанным — посвятив долгие годы разработке аппаратных средств, я на короткой ноге с прикладными научными дисциплинами. Я все же предпочитаю рассматривать нашу отрасль как жанр искусства, хотя, конечно, признаю, что в определенных случаях без строгих научных изысканий не обойтись. Более подробно об этом я поговорю в следующей главе.

[2] Gene Kranz, Failure is Not an Option (New York: Simon & Schuster, 2000), p. 393.

[3] Естественно, при разработке критически важных приложений ставки повышаются. Я до сих пор не доверяю финансовым операциям через Интернет. Знаете, почему? Потому что я знаю тех ребят, которые писали программы для этих операций.

Применимы ли принципы Кранца в нашей области? Думаю, вполне, и мне к ним даже нечего добавить. Это четкие, справедливые принципы, которые неплохо бы записать на бумажке, приклеив ее к монитору. Как я уже неоднократно говорил, совершенствование лидерских качеств повышает шансы на достижение успеха, а в части планирования без лидерства не обойтись. Не стоит сваливать эти обязанности на начальницу — не забывайте, что помимо вашего отдела ей, скорее всего, подчинены несколько других. Представьте себя локомотивом коммерческих достижений компании. Если следовать этой аналогии, получается, что вам, с одной стороны, требуются топливо и грамотная эксплуатация, с другой — кто-то должен жать на газ. Может быть, вы — свеча зажигания? Насколько резво вы искрите? Хватает ли вашей увлеченности, чтобы зажечь искру энтузиазма среди подчиненных?

Знайте свой потолок

Вероятно, в планировании ваша начальница достигла больших успехов, чем вы. В конце концов, почему она оказалась в своей должности? Скорее всего потому, что за ней закрепилась репутация человека, который сочетает в себе навыки планирования и исполнения. Кроме того, вполне возможно, что и во всем остальном она более квалифицирована. В то же время одного она не может знать лучше, чем вы, — вашей верхней планки. Если вы следите за моей мыслью с самого начала книги (надеюсь, что это так), вспомните: в главе 2 мы говорили о том, как преодолевать собственные слабости.

Рассмотрим аналогию из одной программы.

Обычно мы считаем, что в области технологии, инженерии, науки, программирования — да чего бы то ни было! — достигли неплохих успехов. Люди с таким самомнением зачастую ставят перед собой сложные задачи, испытывая тем самым свой интеллектуальный уровень. Я давно играю в шахматы — мне это нравится, хотя свои успехи я оцениваю сдержанно. Мои программные шахматы позволяют устанавливать уровень квалификации противника, роль которого, естественно, исполняет компьютер, — с тем, чтобы любой игрок мог играть в свое удовольствие. Таким образом, если «человеческий» игрок хочет выиграть, он должен заменить максимальный уровень квалификации более низким. Обращаясь к терминологии этой игры[1], назову восемь уровней квалификации:

1. Новичок.
2. Начинающий.
3. Простой.
4. Упрощенный.
5. Средний.
6. Сложный.

---

[1] Chess Master 5000. Не думаю, что производитель будет сильно против, если я позаимствую из его игры структуру меню — в конце концов, это устаревшая версия. Кстати, более свежую и, соответственно, более мощную версию мне одолеть пока что не удалось!

7. Специалист.
8. Чемпион.

Вашу квалификацию по части выполнения рабочих функций, очевидно, тоже можно оценить в границах этого спектра. Квалификация начальницы, скорее всего, выше. Одновременно на рынке идет борьба за звание чемпиона. При общении с начальством вы не должны заблуждаться относительно уровня собственных знаний. Начальница выше оценит честное «не знаю», чем убедительное, на первый взгляд, мнение, которое высказывается по незнанию проблемы.

При общении с начальством вы не должны заблуждаться относительно уровня собственных знаний. Начальница выше оценит честное «не знаю», чем убедительное, на первый взгляд, мнение, которое высказывается по незнанию проблемы.

Раз уж я упомянул шахматы, обратимся к стратегическому и тактическому мышлению — очередным ориентирам, помогающим оценивать квалификацию. В шахматах необходимы навыки стратегического планирования; тактики, как правило, в них не выигрывают — за исключением, конечно, ситуаций, когда оба игрока торопятся закончить игру. В отношениях с начальницей всегда имейте в виду эти два способа решения задач: стратегию и тактику. Скорее всего, в стратегическом отношении она вас превосходит. Если это не так, постарайтесь ей помочь; если же я прав, учитесь! Вероятно, она сможет лучше решать свои задачи, если сконцентрируется на стратегии, а решение тактических вопросов поручит вам. Ваше сотрудничество выгодно обеим сторонам.

Как ожидать неожиданность

Полагаю, ваше положение в компании таково, что стратегические решения принимаются кем-то сверху, а вам остается лишь проводить их в жизнь. В такой ситуации трудно удержаться от ощущения бессилия. И действительно, кому понравится, когда начальство ставит перед фактом очередных решений, которые невозможно было предугадать? По выражению изобретателя современных вакцин Луи Пастер (Louis Pasteur), «шанс приходит только к тем, кто к нему готов». Напишите эти слова на руке или приклейте бумажку с ними на монитор. Наша динамичная индустрия регулярно подбрасывает своим деятелям разного рода сюрпризы. Точно так же любит поступать наше начальство — имейте это в виду и готовьтесь.

Как «подготовиться» к непредсказуемому? Хороший вопрос, на который нет очевидного ответа. С одной стороны, вы должны постоянно пополнять свои знания о технологических возможностях с учетом имеющихся и обещающих вскоре появиться инструментальных средств. Для этого нужно держать в голове технологические новинки в самых разных областях. Эрудитами нам в сегодняшних условиях стать нереально, однако если много читать, разумно пользоваться Интернетом и обращать внимание на конъюнктуру рынка, справиться с поставленной задачей можно. Выделяйте время на ознакомление с информацией, которая может потребоваться в будущем. Не увлекайтесь технологическими игрушками; наблюдайте за новейшими течениями, которым следуют наиболее прогрессивные компании, — если уж ваша организация не относится к их числу, это минимум

того, что вы можете сделать. Обязательно следите за последними бета-версиями широко применяемых в вашей рабочей деятельности инструментальных средств. В идеале, если ваша компания сможет нанять специалиста по исследованию рынка (а возможно, даже создать соответствующий отдел), ваши шансы на адекватную подготовку к будущим изменениям серьезно повысятся. Не стесняйтесь обращаться к помощи начальницы — скорее всего, она готова к изменениям лучше, чем вы. (В разделе «Контролируйте свои слабости» главы 2, в частности, говорится о чтении профессиональной литературы.)

Как преодолеть безынициативность компании

Более чем уверен, что ваша должность относится к среднему звену руководства, а ваша начальница, соответственно, находится в иерархической системе чуть выше. И знаете что? У вас есть преимущество, о котором вы, возможно, даже не подозреваете: вы более восприимчивы к тенденциям в мире разработки программных продуктов за пределами конкретного предприятия. Благодаря своему высокому посту ваша начальница, скорее всего, хуже ориентируется в этих вопросах. Такая ситуация, к сожалению, складывается во многих компаниях, и справиться с ней зачастую нет никакой возможности. Происходит она от чрезмерного внимания, которое менеджеры высшего звена уделяют существующей иерархии. Они тратят массу усилий, чтобы контролировать деятельность нижестоящих сотрудников, упуская при этом ситуацию в индустрии в целом.

Посмотрим, что думает о том, как работают представители менеджмента среднего звена, Энди Гроув:

> «Старшие менеджеры стали таковыми за счет своих профессиональных достижений. Со временем такие люди осознают свои достоинства и, исходя из этого, берут на себя лидерские функции. Таким образом, нет ничего удивительного в том, что и на руководящих постах они продолжают эксплуатировать стратегические и тактические приемы, наработанные в течение карьеры, — в особенности в "чемпионский сезон"»[1].

Гроув называет такое состояние «инерцией успеха»; оно представляет серьезную опасность. Вы, кстати, тоже не застрахованы от этой болезни, так что вещи, которые я рассказываю о вашей начальнице, можете экстраполировать на себя.

Следите за тенденциями в отрасли

Регулярно оценивайте состояние дел вашей компании в контексте рыночной конкуренции. Замечайте действия конкурентов и сравнивайте их с позицией собственной компании. Возможно, ваша начальница настолько погружена в свою текущую деятельность, что даже не допускает мысли о возможности ухудшения положения на рынке. В комплекс обязанностей технического лидера входит, в частности, отслеживание новейших тенденций в индустрии и анализ их влияния на процесс разработки. Если начальница довольна положением дел, это совершенно не значит, что вы должны с ней соглашаться. Если вы видите, что какой-то аспект деятельности компании можно улучшить, подумайте, как это сделать, и донесите

---

[1] Grove, op. cit., p. 127.

ваши соображения до начальства. Для обоснования своих взглядов можете привести ряд примеров из опыта изучения новых методик разработки и анализа вероятности грядущих революций.

Помимо прочего, необходимо следить за признаками пренебрежения со стороны начальства проблемами, с которыми вы сталкиваетесь, — это особенно важно, если компания находится в стадии кризиса. Кризис приходит в высокотехнологичные компании в том случае, если они сначала отказываются повиноваться веяниям времени, а затем внезапно сталкиваются с необходимостью всеобщей модернизации. Выйти из кризиса компания, конечно, должна достойно — это важно и для нее, и для вас. Вы находитесь в уникальном положении — более адекватно воспринимая изменения окружающего мира, вы можете раскрыть глаза начальнице на то, чего она, ослепленная своими былыми успехами, не видит. Естественно, доносить свое мнение нужно осторожно и вежливо, но в то же время уверенно. Вы стоите чуть ли не ближе всех к линии фронта; вовремя сообщать другим представителям компании о допущенных ошибках необходимо — ведь от этого зависят и ваши дальнейшие успехи.

Вы стоите чуть ли не ближе всех к линии фронта; вовремя сообщать другим представителям компании о допущенных ошибках необходимо — ведь от этого зависят и ваши дальнейшие успехи.

Экспериментируйте с новыми методами и приемами

Технический лидер должен апробировать новые методы реализации коммерческих задач. Цель такого рода действий — не в том, чтобы выбрать из нескольких альтернатив. Главное — найти способ снизить издержки разработки и/или повысить ценность предлагаемых компанией продуктов и услуг. Иногда начальство ставит перед нами задачу, детали которой еще только предстоит утрясти. Чувствуете, что инерция берет верх? Попробуйте что-нибудь новое — просто чтобы встряхнуться. Не забывая об осторожности, все же проявляйте лидерские качества: если вы понимаете, что та или иная новация способна улучшить результаты, смело обращайтесь к ней.

Возможно, вашему начальству интересен только конечный результат. Вам, очевидно, тоже, но ведь им проблема не ограничивается! Если за счет внедрения новых методов или инструментов можно сократить срок разработки на 10 %, а издержки — на 20 %, значит, имеет смысл пойти на риск. Если попытка окажется удачной, не забудьте объяснить начальнице, каким образом вы добились успеха. Она оценит старания, ваш статус в ее глазах повысится, и последствия могут быть самыми что ни на есть радужными — как для компании в целом, так и для вас в частности. Это лучший способ стать героем — значительно лучше тех, о которых я говорил ранее.

Учитесь чувствовать время

Корпоративная инерция приводит к тому, что когда экономические или технологические условия диктуют потребность в переменах, сотрудники компании их попросту не замечают. Если ваша начальница знает, что такое успех, вполне вероятно, она чувствует себя неуязвимой и считает, что приспособиться к изменениям

очень просто. Может, она и права, но, в конце концов, это не так уж сложно — следить за признаками перемен.

Не забывайте, что интересы клиента на первом месте

Есть и другой вид инерции, который в технологичных компаниях происходит от чрезмерного внимания к методам реализации в ущерб результату. В конце концов, вы призваны создавать продукты и услуги, которые удовлетворяют потребностям клиентов. Посмотрим, что думает по поводу поддержки клиентов один из лидеров нашей индустрии.

> «Лучшая поддержка клиентов — это отсутствие поддержки клиентов. Да-да, все правильно, потребности в хорошей поддержке клиентов просто не должно возникать. Цель должна быть такой: шаг за шагом совершенствовать продукты и услуги, чтобы клиент без посторонней помощи легко мог их использовать и в них разобраться. Ориентированные на клиента фирмы, если они хотят добиться успеха, должны реализовывать такие технологии, которые позволяют глубже понять потребности клиентов. В результате такой практики компания получает возможность выпускать продукты и услуги, действительно необходимые клиентам. Сообщаю всем зацикленным на окупаемости инвестиций: начальные капиталовложения в поддержку клиентов позволяют впоследствии существенно снизить издержки на поддержку продуктов (чему способствуют сбор и анализ информации о клиентах)»[1].

Если упустить из виду признаки технологической инерции, вы рискуете нарваться на ситуацию, когда из-за пренебрежения к потребностям пользователей после выхода на рынок последнего «чудесного» продукта на корпоративную службу поддержки клиентов обрушится шквал звонков. Чрезмерное увлечение инструментарием чревато потерей внимания к конечным пользователям. Современные технологии крайне сложны, но в то же время они призваны избавить пользователей от лицезрения деталей реализации.

В период планирования следующего продукта попросите начальницу предоставить вам более свободный доступ к маркетинговым данным. Если она отнесется к вашему предложению легкомысленно, обратите ее внимание на необходимость анализа рыночной конъюнктуры — в конце концов, именно для этого существуют отделы продаж. Я уже говорил о том, что сроки сдачи продуктов нередко обусловливаются маркетинговыми соображениями — но, с другой стороны, этот фактор носит прогрессивный характер. Анализ рыночной ситуации существенно расширяет охват аудитории, на который в качестве коммерческого решения может претендовать тот или иной продукт (или услуга). Прошлогодние соображения по поводу требований рынка бесполезны. Сведения о потребностях, которые пользователи испытывают сегодня и будут испытывать завтра, должны быть такими же свежими, как и технологии, позволяющие их реализовать.

---

[1] См. статью Челлис Ходж (Challis Hodge) «Smoothing the Path» по адресу http://www.webtechniques.com/archives/2001/11/hodge/. Ходж — основатель и генеральный директор HannaHodge, — чикагскойкомпании, занимающейся изучением взаимодействия пользователей с системами. Кроме того, он разработал план преподавания дисциплины «проектирование интерфейсов» для Висконсинского университета и возглавил разработку пользовательских интерфейсов для корпоративных веб-решений компании IBM.

Резюме

Итак, я упомянул о некоторых областях взаимоотношений с начальством, на которые вам надлежит обратить особое внимание. Основная ваша цель, по-моему, должна состоять в том, чтобы наладить с боссом конструктивное взаимодействие, от которого выиграют обе стороны. Ниже я привожу краткий перечень наиболее важных моментов этой стороны вашей деятельности.

- Войдите в положение начальницы — она в своей работе подвергается серьезному давлению. Первой реакцией на ее замечания должно быть согласие, однако если у вас есть разумные и обоснованные предложения, выскажите их.
- Честность в суждениях в любом случае предпочтительнее нереалистичных обещаний — пусть даже она иногда приводит к не слишком приятным последствиям.
- Помогайте начальнице в вопросах планирования. Настаивайте на необходимости разработки реалистичных планов и прилагайте максимум усилий к их реализации.
- Если вам нечего противопоставить замечаниям начальницы, признайте поражение. Не стоит вешать ей лапшу на уши в попытках замаскировать собственные огрехи.
- Будьте готовы к непредвиденным изменениям — чем лучше вы будете осведомлены, тем проще будет с ними столкнуться. Иногда неожиданные решения исходят от начальства, в иных случаях — обусловливаются внешними факторами. Тщательная подготовка есть залог ваших успехов[1].
- Следите за признаками инерции в самом себе и в начальнице. Если она отрицает очевидную потребность в изменениях, старайтесь спасти положение, пока не поздно.

Чтобы окончательно закрыть тему взаимоотношений с начальницей, скажу еще одну вещь. Обращайтесь с ней так, как хотели бы, чтобы обращались с вами. Это золотое правило, которое, хоть и ассоциируется с воскресной школой, работает во всех сферах человеческой деятельности.

Конец уже близко

Не карьеры, а книги. Следующая глава будет последней. В ней рассматриваются самые разные вопросы, которым в предыдущих главах было уделено недостаточно внимания или которые вообще не упоминались. Материал книги я решил структурировать по тематическому принципу — в соответствии с различными областями деятельности, с которыми менеджеру приходится сталкиваться каждый божий день. Следующая же глава, по большому счету, бессистемна. В ней вы, в частности, познакомитесь с необычными для руководителя программистов проблемами, выходящими за рамки повседневной деятельности.

---

[1] Это «переработка» известного высказывания Пастера (Pasteur): «шанс приходит только к тем, кто к нему готов». Авторство парафраза принадлежит не мне — впервые я его услышал в детстве из уст отца.

Глава 10
Слова без песни

В целом разработка программных продуктов носит линейный характер. Это проявляется даже тогда, когда применяется итерационный процесс. В любом случае переход от одного этапа к другому логичен. Руководство процессом разработки, напротив, представляет собой нелинейную область деятельности. Вы постоянно вынуждены переходить от одной проблемной области к другой, причем решения, имеющие ценность в одной из таких областей, могут не иметь никакого смысла в любой другой. Таким образом, лучшее, что вы можете сделать, — это привлечь весь свой интеллектуальный ресурс к решению текущих проблем, не забывая при этом про конечную цель: вывести подчиненных из состояния хаоса и заставить их двигаться в одном направлении. Такую стратегию, наверное, можно обозначить как «жизнь над краем пропасти» — думаю, именно в таком состоянии вы проводите многие дни на работе.

Название данной главы выражает вот это самое состояние. Отчасти это перекличка с традицией композиторов эпохи романтизма, которые широко эксплуатировали жанр песни без слов (к таковым, в частности, относится Мендельсон). Мне кажется, из всего классического репертуара эти композиции — чуть ли не самые мелодичные. Смысл этой главы, по большому счету, в обратном: общая тема отсутствует, а отдельные партии довольно сложно назвать гармоничными. Может показаться, что в этой главе я собрал все темы, которые по тем или иным причинам не нашли отражения в предыдущих главах, — и это ощущение правильно. Впрочем, такую бессистемность можно обосновать и по-другому, более хитро — дело в том, что иногда наша работа производит впечатление совокупности ничем не связанных действий. Бывает, нам даже приходится разбираться с проблемами, способы решения которых в предыдущем опыте отсутствуют. Вот в этом направлении я и выстроил главу[1]. Надеюсь, вопросы, которые в ней поднимаются, внесут некоторую ясность в ваши обязанности.

---

[1] Помимо всего прочего, я опасался, что глава под названием «Разное» не вызовет должного читательского интереса.

Распределенная рабочая сила

Современная корпоративная культура зачастую не оставляет места для географической централизации групп разработчиков. Финансовые соображения, наличие выдающихся особо талантливых специалистов, равно как и традиции конкретной компании, — все это иногда приводит к тому, что ваши подчиненные, хотя и образуют единую группу, фактически работают в разных местах. Если сотрудники находятся в разных зданиях или даже в разных часовых поясах, обеспечить гармоничное сочетание сотрудничества с уединением, что является основным фактором продуктивной деятельности группы, не так-то просто[1]. Если среди ваших подчиненных есть такие, которые один или два раза в неделю общаются с остальными на телеконференциях, вам не обойтись без специальных методов, позволяющих контролировать их продуктивность.

Если сотрудники находятся в разных зданиях или даже в разных часовых поясах, обеспечить гармоничное сочетание сотрудничества с уединением, что является основным фактором продуктивной деятельности группы, не так-то просто.

Суть проблемы

Если вам удается успешно сочетать сотрудничество с уединением, процесс разработки существенно упрощается.

- *Уединение*. Способность работать, не отвлекаясь на внешние раздражители, в условиях, допускающих абстрактное мышление.
- *Сотрудничество*. Обмен предложениями на очной встрече вплоть до выработки консенсуса во мнениях всей группы или нескольких программистов, работающих над решением одной задачи.

Группа, в которой удовлетворяются две вышеупомянутые потребности, способна работать крайне продуктивно. Для написания качественного кода необходим баланс между этими условиями.

В контексте распределенных групп, в которых одни сотрудники работают дома, другие трудятся в офисе вместе с остальными не связанными с ними функциональными обязанностями специалистами, третьи образуют более мелкие группы, также работающие удаленно, обеспечить сотрудничество и уединение довольно трудно. Одни специалисты, не испытывающие недостатка в уединении, будут лишены средств продуктивного сотрудничества. Другие могут, с одной стороны, испытывать нехватку уединения, а с другой — активно общаться с чужими с точки зрения выполняемых обязанностей людьми. Проблемы, которые возникают в подобных ситуациях, можно обобщенно сформулировать следующим образом.

- Решения, которые при условии географической централизации принимаются за считанные минуты, растягиваются на многие часы и даже дни.
- Непосредственная проверка деятельности и продуктивности персонала заменяется взаимодействием с сотрудниками по телефону и электронной почте.

[1] О сотрудничестве и уединении мы говорили в главе 4.

Таким образом, хрестоматийный принцип руководства «доверяй, но проверяй» не соблюдается.

- Техническое проектирование осуществляется в основном не в интерактивном режиме, а посредством документов. Документы же должны быть не средством, а результатом сотрудничества. Из-за ограничений по времени этап проектирования растягивается или вообще пропускается.
- Сотрудники группы лишены чувства работы в единой атмосфере. Возможности для выстраивания дружеских отношений отсутствуют.
- Виртуальное рабочее пространство формируется средствами электронной почты и службы мгновенной передачи сообщений. При всей своей полезности они не заменяют очного взаимодействия.
- Большую часть времени вы, руководитель, проводите с телефонной трубкой в руках.

Некоторые из вышеупомянутых проблем проявляют себя даже при условии географической централизации, однако руководитель, вынужденный координировать действия сотрудников, работающих в разных местах, сталкивается с ними в наихудших проявлениях.

Решение

Если провести централизацию нет никакой возможности, значит, придется адаптировать ваш стиль руководства к создавшимся условиям. Как я говорил в главе 1, адаптация — это тот навык, без которого руководителю программистов не обойтись. Рассмотрим, как он проявляется в некоторых аспектах деятельности руководителя в условиях распределенной группы.

Планирование

Во-первых, вы должны тщательно продумать вопрос распределения задач. Это невозможно без предварительного планирования проекта, позволяющего выделить в его рамках отдельные блоки, которые можно распределить между подчиненными с поправкой на недостаток сотрудничества. Конечно, лучше всего детально планировать все проекты, но в случае с распределенной группой эта проблема приобретает особую актуальность.

Это невозможно без предварительного планирования проекта, позволяющего выделить в его рамках отдельные блоки, которые можно распределить между подчиненными с поправкой на недостаток сотрудничества.

Если вам не удастся четко сформулировать отдельные задания, на их уточнение уйдет гораздо больше времени, чем обычно, — просто в силу удаленности от сотрудников. Кроме того, вы должны спланировать деятельность персонала таким образом, чтобы результаты его работы легко стыковались. В таких условиях предпочтение нужно отдавать компонентной архитектуре — если, конечно, она вписывается в корпоративную стратегию. Конструирование программных продуктов, таким образом, должно походить на сборочный конвейер, на котором каждая деталь идеально стыкуется со всеми остальными. Хороший принцип, который

неплохо было бы реализовать. Это вполне возможно, хотя и требует значительных усилий по части планирования и проектирования.

Вообще-то планирование, проводящееся на ранних этапах разработки любого проекта, предполагает очное взаимодействие. Иначе говоря, чтобы приступить к созданию проектного решения, по крайней мере один раз вам придется собрать всех сотрудников в одном месте. Это недешево, но временные затраты на проектирование по электронной почте или по телефону оказываются значительно серьезнее. Постарайтесь во время этого сеанса планирования решить две задачи: утвердить основы проектного решения и сделать первые шаги в направлении формирования команды. Полезно в этом смысле где-то раз в год выезжать вместе с сотрудниками на отдых в экзотические страны. Впрочем, вполне возможно, что, узнав цены на авиабилеты «туда и обратно», вы решите, что куда мудрее будет развернуть за те же деньги каналы для проведения видеоконференций или, например, заказать какую-нибудь хитрую систему, которая могла бы отображать электронное табло на одном мониторе и картинку нескольких групп на другом. Конечно, все подобные экономические вопросы нужно решать исходя из бюджета. В любом случае, пользуясь имеющимися у компании средствами, вы должны поощрять взаимодействие участников группы.

Взаимодействие

Большое человеческое спасибо Александру Грэму Беллу (Alexander Graham Bell)! Телефон значительно облегчил для нас двустороннюю связь в реальном времени. Правда, полагаю, что Белл воскликнул «Поди-ка сюда, Ватсон!», просто утомившись от своего аппарата и решив, что лучше переговорить с помощником с глазу на глаз. Если бы все ваши подчиненные находились в одно время в одном месте, планировать распорядок дня было бы гораздо проще. Тем не менее, если, конечно, вы не сводите все взаимодействие к общению по электронной почте, имейте в виду, что в условиях децентрализации группы телефон представляет собой заметную ценность. Если вы можете себе позволить организацию регулярных видеоконференций, качество взаимодействия, несомненно, повысится. Впрочем, далеко не все компании на это способны. При наличии канала с достаточной пропускной способностью можно разориться на веб-камеры и соответствующее программное обеспечение. Замечу, что с этими средствами мне не удалось добиться сколько-нибудь заметных успехов.

Естественно, в большинстве географически рассредоточенных групп электронная почта выступает в роли основного средства взаимодействия. Для менеджера в практическом смысле это означает, что на проверку новых писем приходится тратить значительную часть рабочего дня. Но ведь у вас много других административных обязанностей, а значит, нужно учиться отвечать на письма как можно быстрее. Попытайтесь ограничить переписку по электронной почте темами, которые можно адекватно выразить в письменной речи или на схемах. Все дискуссии следует проводить по телефону, а для решений, требующих документирования, вполне подойдет электронная почта.

Невзирая на все недостатки документно-ориентированного процесса проектирования, скорее всего, именно на нем вы остановите свой выбор. Кроме того, полезен в данном контексте метод проектирования по контракту, подразумеваю-

щий первоочередное (в самом начале периода проектирования) создание открытых интерфейсов объектов, с тем чтобы впоследствии их можно было без труда разработать. Этот метод полезен даже в условиях централизации группы, а уж если один разработчик трудится в Калифорнии, а другой — где-нибудь в Новой Англии, соглашение по части взаимодействия компонентов и их способности к взаимодействию выходит на первый план. Купите качественное программное обеспечение для совместной работы и задействуйте его в роли библиотеки документов. За рыночную нишу программных продуктов для совместной работы соревнуются несколько производителей, и это вам на руку — следите за их последними разработками[1].

Проверка

В том что касается проверки результатов сотрудников децентрализованной группы, ваши обязанности весьма осложняются. Из-за разницы временных поясов вам, вероятнее всего, придется удлинить свой рабочий день. Такое решение — фактически единственный способ эффективной организации в данных условиях — требует от вас выносливости. Если вам приходится ежедневно проводить на работе по 12 часов, не забывайте о том, что вы живой человек, и кроме профессии в жизни есть еще кое-что. Рекомендую взять за правило при первой возможности делать перерывы и посвящать их личным делам — в противном случае вы вряд ли долго протянете.

Удаленный мониторинг персонала — дело ненадежное. Бывает, проходишь за спиной сотрудника, который вместо работы шатается по Сети, и спрашиваешь: «А почему не работаешь?!» Ну, он, конечно, отвечает: «Знал бы, что вы рядом, принялся бы за работу!» Ситуация, на первый взгляд, забавная, удачно характеризует особенности роли «виртуального шефа». В отсутствие начальника люди начинают валять дурака — это в природе человека. Как с этим бороться? Сравнительно эффективный мониторинг возможен только при наличии формального плана сдачи сотрудниками результатов и при условии тщательной проверки его выполнения. Структура распределения работ в данном случае должна быть мельче, чем обычно, — так вы сможете с большей уверенностью предотвращать отклонения от графика разработки проекта. Для выполнения контрольной функции вам опять же понадобятся телефон и электронная почта. Заставьте сотрудников присылать еженедельные отчеты (а в исключительных случаях — даже ежедневные).

Можно также обратиться к решению в духе Оруэлла — оснастить рабочие станции разработчиков программами дистанционного мониторинга, позволяющими в любой момент видеть их экран. Сами сотрудники, естественно, должны знать, что их компьютеры «прослушиваются», и если уж необходимость плотного слежения за каким-то деятелем назрела, это решение себя оправдывает. Правда, надо сказать, оно совсем не способствует укреплению доверительных отношений — скорее, напротив, вызывает негодование. Поэтому прежде чем прибегать к подобным мерам, хорошо подумайте — стоит ли.

---

[1] См. http://www.eroom.com, http://www.fox.se/english/starteam/starteam_version_control.htm и другие предложения от IBM и Symantec.

Сравнительно эффективное отслеживание возможно только при наличии формального плана сдачи сотрудниками результатов и при условии тщательной проверки его выполнения.

Личные контакты

Чем чаще вы будете посещать своих сотрудников в их привычных рабочих условиях, тем выше шансы на построение полноценной команды. Обозначая свое присутствие на рабочем месте, вы способствуете развитию у сотрудника чувства участия в коллективной деятельности и придаете вес своему лидерству в остальное время. На подобные визиты уходит много времени, да и не бесплатно это, однако таким способом вы сможете смягчить некоторые недостатки децентрализации.

Культурный фактор в менеджменте

Америка — настоящий плавильный котел народов мира; нет — точнее, наверное, будет сказать «культурный винегрет». И, надо заметить, программировать умеют не только американцы. Научиться руководить сотрудниками, относящими себя к разным национальностям и культурам, не так уж просто. Имея дело с неоднородным в культурном отношении персоналом, вы должны в первую очередь видоизменить привычные методы взаимодействия и мотивирования. Стремитесь к неофициальному общению с представителями иных народов — интересуйтесь их семейными обычаями и культурными традициями. Так вы сможете добиться от них большего рвения; кроме того, видя, что вы пытаетесь вникнуть в их особое миропонимание, программист, скажем, из России [где это?] сможет почувствовать свою ценность в команде.

Язык и культура

Работая с представителями разных культур, вы должны привести свою речь к определенному стандарту. С чужаками нельзя общаться так же, как с корешами из соседнего двора, — не выйдет! Структуры речи, общераспространенные разговорные стили и даже язык тела — во всех этих отношениях между разными культурами наблюдаются серьезные различия. Заставьте себя говорить медленнее — поймите, что факт употребления английского как второго языка еще не означает знания американского сленга, адекватного восприятия привычки к небрежному построению предложений и следования вашему ходу мысли, выраженному в словах. Даже британский английский в некоторых моментах существенно (и радикально!) отличается от американского[1].

Нельзя общаться с чужаками так же, как с корешами из соседнего двора, — не выйдет!

[1] Не говорите англичанину про богатый (rich) пользовательский интерфейс. Он обязательно подумает, что интерфейс слишком дорогостоящий.

Есть и другие культурные различия, которые иногда удивляют, а иногда кажутся достойными включения в стандартную корпоративную практику. Знаете ли вы, например, что среди филиппинцев публичное рукопожатие служит признаком дружбы? Когда индиец первый раз приходит в гости, он обязательно приносит подарок. Австралийская культура, на первый взгляд, во многом пересекается с американской, но знаете ли вы, в каких количествах австралийцы поедают овощную пасту?[2] А как насчет традиционных американских праздников? Уверяю вас: человек, родившийся в Бангалоре, в День благодарения будет, как обычно, работать. Вы должны выяснить все существующие различия и стать гражданином мира — усваивайте лучшие черты всех культур и включайте их в свой «джентльменский набор» лидера.

Мотивация чужаков и контроль над ними

Мотивация программистов деньгами среди стандартного набора американских приемов менеджмента является «наименьшим общим кратным». С другой стороны, имейте в виду, что деньги — это наименее эффективный способ построения командной атмосферы в многокультурной среде. Что же делать? Проанализировать все факторы, способные стимулировать к работе существующую команду или новых сотрудников, которых вы только собираетесь привлечь. По результатам недавнего опроса 100 наемных работников, не являющихся гражданами США, Дин Бетменг (Dene Bettmeng) делает следующие выводы.

> «Многие сотрудники рассматривают в качестве основных преимуществ работы в американских компаниях корпоративную культуру и корпоративные правила, а также технологическую оснащенность. В частности, их привлекают перспективы карьерного роста, заработная плата и льготы, равно как и возможность изучать и применять новые технологии. Все эти люди считают, что работать в компании, расположенной в США, в целом предпочтительнее, чем в какой бы то ни было другой стране. Помимо вышеперечисленных преимуществ, они ценят политическую стабильность, напряженный ритм деятельности и командный дух»[2].

Как и в любой команде, руководителю необходимо составить представление о своих разношерстных подчиненных — иначе он не сможет преуспеть в качестве лидера. Наличие в подведомственной группе представителей различных культур усложняет познание, но в то же время дает неоценимый (в зависимости от ситуации, положительный или отрицательный) личный опыт. Сталкиваясь с трудностями в поисках свежих американских талантов, мы обращаем внимание на заморских консультантов. Иной раз ставка на них себя оправдывает, но бывает и по-другому. В некоторых культурах споры с начальником (или даже уточнение его позиции) не поощряются. В результате начальник утверждается во мнении, что подчиненный понял, чего от него хотят, а в том, как жестоко ошибся, убеждается, когда уже слишком поздно. В этой области нужно соблюдать осторожность.

---

[1] Хотя она чем-то напоминает арахисовое масло, уверяю вас, совсем не так вкусна!

[2] Из отчета на конференции Network World Fusion (http://www.nwfusion.com/careers/2001/0402man.html).

КОШАЧЬИ РАЗБОРКИ — ИНОСТРАННЫЙ ЛЕГИОН

Мы решили как можно быстрее заполнить вакансии в трех проектах за счет привлечения 12 иностранных консультантов. Кандидаты ниже экспертного уровня нас не интересовали — по крайней мере, именно таким образом мы сформулировали свои требования. Как выяснилось, все специалисты, которых нам подобрали, приехали из Индии — страны, стремительно укрепляющей свои позиции по части квалификации программистов. Мне предстояло составить из них группы в соответствии со специальностями и приступить к работе над проектами. Я участвовал во всех интервью, отбирая кандидатов по специальности. К сожалению, все интервью проводились по телефону — тут-то я и получил первый урок. Естественно, осознать, какую головную боль я себе организовал, мне удалось лишь тогда, когда сформированные группы принялись за работу.

Урок 1. Впоследствии выяснилось, что для телефонных интервью консультантам предоставили длиннющие шпаргалки с ответами на все мыслимые и немыслимые вопросы. Стоило мне спросить: «Какое средство моделирования данных вы предпочитаете?» — как абонент, выдержав паузу, переспрашивал: «Повторите, пожалуйста, вопрос — я не совсем понял, о чем речь». В это время он листал шпаргалки в поисках правильного ответа. Отыскав его, он отвечал: «Больше всего мне нравится ERStudio, но у него есть такие-то недостатки. ERWin — тоже ничего, он выгодно отличается тем-то и тем-то». Понимаете, дословно повторяя содержимое шпаргалок, они говорили все то, что я хотел услышать. Мне уже стало казаться, что группа будет состоять исключительно из гениев. Вскоре после начала работы над проектом я понял, как сильно ошибался.

Собственно, по мере продвижения проектов я получил еще несколько печальных уроков. Все мои неудачи происходили от того, что я совершенно не учитывал культурные факторы — я даже не предполагал, что в деле управления группами, состоящими из зарубежных специалистов, культурные факторы имеют какое-то значение.

Урок 2. Когда индиец кивает головой, он имеет в виду «нет», а не «да». Вы можете себе представить, как это сбивает с толку?! Ведь мы принимаем за аксиому, что кивок выражает согласие!

Урок 3. В большинстве случаев специалисты из Индии не препираются, не пытаются вас поправить и не подвергают сомнению авторитет начальства — по крайней мере публично. Таким образом, они согласятся со всем, что вы скажете, — пусть даже вы будете нести полную чушь. Некоторые из них после совещаний продолжают решать задачу по-своему, вне зависимости от того, что им сказали. Работая с индийцами, я возомнил себя великолепным лидером — ведь американские программисты ни за что не позволят достичь консенсуса так быстро и легко!

Урок 4. Как правило, если консультанты не понимали, что я им говорил, они даже не пытались дать об этом знать. Они несколько раз повторяли, что все мои предложения в высшей степени разумны, после чего продолжали кодировать так, как считали нужным.

Урок 5. Последняя индийская перепись населения зафиксировала в стране более 200 языков и диалектов. Поскольку все мои сотрудники приехали из разных регионов Индии (кстати сказать, друг друга они недолюбливали), между собой они общались исключительно на ломаном английском. Многочисленные улыбки и жесты отнюдь не способствовали преодолению взаимного непонимания.

Урок 6. Несмотря на то что всех новоиспеченных сотрудников нам привело одно и то же агентство, все они работали на разные консалтинговые компании. Я платил за их услуги по 125 долларов в час, из которых самим исполнителям доставалось только от 15 до 55.

Но... все эти уроки я выучил слишком поздно. Лишь один из трех проектов был успешно завершен; остальные два постигла печальная судьба — немалые деньги оказались выброшенными. Ни за что больше не буду ввязываться в подобные авантюры.

Оценка методологий разработки программных средств

Раньше программирование считалось уделом инженеров и ученых, весьма далеких от суеты бизнес-центров. Нынче компьютерами пользуются все подряд, а употребление технологических достижений в коммерческих целях обязывает создавать качественные программные продукты вовремя и в рамках установленного бюджета. К последнему утверждению можно, конечно, относиться с иронией, но факт остается фактом: эпоха программ, разрабатывавшихся энтузиастами по собственным правилам, ушла в историю. Бизнес требует тщательности в разработке и внимания к практическому результату деятельности персонала. Здесь-то и начинается самое интересное. Практически все представители нашей индустрии, получившие какое-никакое признание, считают себя экспертами и норовят продать «уникальные» методологии разработки.

Со мной другая ситуация — я пытаюсь продать принцип эклектичности в методиках разработки. Задействуйте и совершенствуйте те методы, которые доказали свою практичность для вас лично, для группы и для компании. Создавайте собственные методы и приспосабливайте их для нужд своей организации — вне зависимости от того, что они собой представляют, суть должна быть одна: оперативная разработка качественного программного обеспечения. В следующих разделах я привожу обзор заметных школ разработки программных средств, причем основной акцент делаю на их концептуальной основе. Школы проектирования я обсуждать не собираюсь. Структурное программирование, объектно-ориентированное проектирование, эталоны проектирования — все эти течения в основном сфокусированы на деталях конструкции и значительно меньше внимания уделяют общей методологии разработки. Архитектурные проблемы (рассматриваемые в главе 6) также не получают в них достаточно глубокого развития. Итак, отступив от деталей, я собираюсь представить на ваш суд наиболее общий анализ ситуации.

Создавайте собственные методы и приспосабливайте их для нужд своей организации — вне зависимости от того, что они собой представляют, суть должна быть одна: оперативная разработка качественного программного обеспечения.

Программная инженерия

В конце 1960-х годов, когда тогдашним инженерным гениям удалось высадить человека на Луну, располагая при этом меньшими вычислительными мощностями, чем любой современный калькулятор[1], в обиход вошел термин «программная инженерия». Эта концепция зиждется на представлении, согласно которому программное обеспечение можно разрабатывать средствами традиционных инженерных дисциплин. Применительно к сверхкрупным программным проектам, которые,

[1] Первый функционирующий микропроцессор появился в 1971 году.

как правило, заказывались правительством или крупными подрядчиками министерства обороны, эта методика несколько раз показала достойные результаты, но в остальных случаях потерпела фиаско[1]. Метод программной инженерии зачастую применялся в условиях одновременной разработки программного и аппаратного обеспечения. Кроме того, с его помощью удалось создать несколько коммерческих продуктов. IEEE определяет этот метод следующим образом:

> «Программная инженерия — это систематический, структурированный, количественный подход к разработке, функционированию и сопровождению программных средств; иначе говоря, способ применения инженерных методов в разработке программных продуктов»[2].

Полагаю, вы, как и я, в замешательстве. Основное внимание в этом методе уделяется сокращению количества дефектов в коде — о соблюдении бюджетных рамок ни слова.

Это не в меру сжатое определение, по-моему, можно расширить.

- *Систематически* — все аспекты разработки можно контролировать в едином процессе.
- *Структурно* — последовательное употребление должных методов с целью производства качественного программного продукта.
- *Количественно* — все требования известны и допускают отображение на методы реализации.

Прежде чем чесать затылок, глядя на мою интерпретацию трех основных концепций программной инженерии, согласитесь: всем нам хотелось бы достичь определенности, точности и завершенности, которые декларируются. Посмотрим, как относится к мифу о суперпрограммистах один из наиболее последовательных поборников программной инженерии:

> «Существует распространенное мнение, согласно которому несколько высококлассных специалистов, собранных вместе, способны сделать больше, чем стандартная группа разработчиков. Имеется в виду, что их знания о способах достижения высоких результатов интуитивны, а значит, потребность в систематизации процесса отпадает. Будь это действительно так, можно было бы предположить, что компании, у которых в штате числятся наиболее квалифицированные специалисты, не должны испытывать типичных проблем с качеством разрабатываемых программных средств и продуктивностью. Опыт же подсказывает нам, что это совершенно не так»[3].

Видите, какой получается конфликт. «Человек» противопоставляется «машине» — в контексте программной инженерии «машиной» можно назвать дисциплинированную группу разработчиков, работающую на основе общей методологии. «Человеком» же обозначается программист-светило, который как по волшебству мастерит продукты, руководствуясь исключительно своими познаниями. В дискуссию по этому поводу (по-моему, она немного искусственна) вступать не стоит. Хорошие специалисты и надежный процесс в равной степени необходимы — не

---

[1] См. Glass, Software Runaways, op. cit.

[2] См. IEEE Standard Computer Dictionary (Institute of Electrical and Electronics Engineers, Институт инженеров по электротехнике и электронике, 1990).

[3] Humphrey, op. cit., p. ix.

будь одного из этих компонентов, создавать качественные программные продукты было бы крайне сложно. Вопрос лишь в том, является ли надежным процессом программная инженерия.

Мнения на этот счет расходятся. Предположим, перед вами стоит задача конструирования системы управления воздушным движением. На первый взгляд, программная инженерия прекрасно подходит для ее решения. Но... стоит подумать еще раз. К провалу проекта комплексной системы автоматизации (Advanced Automation System, AAS) Федерального управления гражданской авиации США (Federal Aviation Administration, FAA), с помощью которого в 1980-х годах предполагалось модернизировать систему управления воздушным движением, методика программной инженерии имеет непосредственное отношение. Несмотря на миллионные затраты, проект так и не оправдал ожиданий[1].

Кое-кто до сих пор убеждает общественность в том, что программная инженерия — это *именно то, что нужно*; их оппоненты не соглашаются — по их мнению, эта методология оторвана от реальности и с наступлением эпохи Интернета неприменима к рутинным проектам. Я предлагаю вам самостоятельно изучить соответствующую литературу, опробовать на практике декларируемые в ней идеи и лишь после этого делать выводы. В рамках школы программной инженерии исследователям удалось выработать ряд весьма удачных идей, и если они вам подходят — действуйте! С моей точки зрения, программная инженерия — это скорее идеальная конструкция, чем реальная методология; по этой причине методы для внедрения в повседневную деятельность своей команды я предпочитаю искать в других концепциях.

MSF

Хотите — возмущайтесь, но в одном вы меня не переубедите: компании Microsoft удалось разработать крайне продуманный метод и успешно его разрекламировать. Несколько компаний[2] даже решили в приказном порядке отправить своих сотрудников на недельные курсы обучения предлагаемому Microsoft методу. Не сомневаюсь, что компании уровня Microsoft, специализирующейся на разработке коммерческих программных продуктов, есть что сказать на заданную тему.

Концептуальная основа метода MSF (Microsoft Solutions Framework — каркас решений Microsoft) предполагает координацию групп, тем или иным способом ответственных за процесс разработки. Вместо того чтобы воспроизводить яркие рекламные лозунги, я сосредоточу ваше внимание на отдельных деталях метода — по-моему, они лучше любых сообщений информационных служб и даже лучше научного изложения концепции отражают позицию Microsoft. Согласно выпущенному компанией руководству[3], предлагаемая методика основывается на «трех китах».

1. Прикладная модель строится на основе предоставляемых услуг, побуждая разработчиков рассматривать любое приложение как сеть услуг, в которой характеристики и функциональность можно пакетировать вне зависимости от функциональных границ и в расчете на многократное использование.

---

[1] См. Glass, Software Runaways, op. cit., p. 56.

[2] В том числе и моя, естественно.

[3] Solutions Development Discipline Workbook (Redmond, WA: Microsoft Corporation, 1996), pp. 1–17.

2. Итерационная модель процесса жизненного цикла разработки ориентирована на поставки, отталкивается от рисков и включает четыре основных этапа.
3. Используется модель масштабируемой группы разработчиков, состоящей из шести равнозначных ролей.

Не сомневаюсь, что вы с нетерпением ожидаете изложения упомянутых «четырех этапов» и «шести ролей». Сходите на курсы Microsoft — я не хочу рекламировать специалистов Microsoft больше, чем они того заслуживают. Они высказали ряд замечательных идей, которые мне удалось приспособить к своей методологии разработки. Ну ладно — полагаю, у вас не так много времени, чтобы проводить собственные исследования, так что, коль скоро вы купили мою книгу, поговорим о деталях.

Что касается этапов MSF, то здесь группа разработчиков должна принять на вооружение ряд принципов.

1. *Видение/рамки*. Основное внимание уделяется не требованиям, а рамкам работ.
2. *План проекта*. Заказчики и участники группы должны договориться о поставках, приоритетах и ожиданиях.
3. *Закрытые рамки/Первое применение*. Первая бета-версия готового продукта.
4. *Выпуск*. Продукт или услуга предоставляется рабочей группе и группе поддержки.

Роли в MSF распределяются следующим образом.

1. *Руководство продуктом*. Особое внимание уделяется оценке продукта с коммерческой точки зрения.
2. *Управление программой*. Разработка функциональных спецификаций, утверждение и корректировка графика.
3. *Разработка*. Конструирование продукта (или услуги), соответствующего спецификации и ожиданиям заказчиков.
4. *Тестирование*. Выявление всех проблем перед выпуском программы.
5. *Обучение пользователей*. Каждый пользователь должен получать от продукта максимум возможностей.
6. *Логистика*. Обеспечение беспрепятственных выпуска, установки и миграции.

Все очень складно, не правда ли? На самом деле максимальной продуктивности, вооружившись методикой MSF, можно достичь лишь при наличии достаточно многочисленного персонала, который позволит сформировать группы и исполнять процессы согласно рекомендациям. Преподаватели на курсах утверждают, что метод MSF применяется в самой компании Microsoft. Учитывая характер литературы, выпущенной Microsoft Press за последние 10 лет, полагаю, что это действительно так[1].

Вам лишь остается выяснить, насколько успешно методология MSF может проявить себя в условиях конкретной группы и компании. Она ведь не ограничива-

---

[1] В первую очередь, я имею в виду работу Jim McCarthy. Dynamics of Software Development (Microsoft Press, 1995). Вероятно, именно в ней впервые сформулированы некоторые основополагающие принципы MSF.

ется одной разработкой. В ней предпринимается попытка направить усилия всего предприятия на достижение одной цели — поставки достойного по качеству программного обеспечения. Лично я положительно оцениваю свой опыт применения MSF — правда, этот метод требует адаптации к реалиям корпоративной культуры и корректировки отдельных характеристик рекомендованных групп.

Экстремальное программирование

С одной стороны, экстремальное программирование (Extreme Programming, XP) позиционируется как новаторская методика; с другой — эта методика существует, пожалуй, с тех давних пор, когда в реле находили тараканов[1]. Позвольте пояснить. Основное внимание в XP уделяется командному программированию с акцентом на код сам по себе. Последний понимается как средство передачи требований и изменений, а также ключ к пониманию задачи программного продукта. Один из ведущих проповедников XP Кент Бек (Kent Beck) утверждает, что XP обещает радужные перспективы двум группам заинтересованных лиц:

> «Программистам XP предоставляет возможность каждый день сосредоточиваться только на тех вопросах, которые действительно важны. Им больше не придется сталкиваться с устрашающими ситуациями один на один. Они смогут проявить свои способности на все сто, направить их всецело на обеспечение качества системы. Им не придется принимать решения в тех областях, в которых они не слишком много понимают, — они могут сосредоточиться на своих прямых обязанностях».
>
> «При помощи XP заказчики и руководители смогут каждую неделю разработки получать максимальный результат. С регулярностью в несколько недель они будут оценивать конкретный результат работы над решением задач, которые для них наиболее важны. Корректировать направление разработки проекта можно будет в самый разгар трудовой деятельности программистов, причем никаких сверхъестественных затрат на это не потребуется»[2].

Каким образом метод XP реализует эти перспективы? Путем введения ряда принципов программирования, которым группы должны следовать неукоснительно. Вместо того чтобы перекладывать отдельные аспекты разработки на руководящее звено, метод XP концентрируется на функциях собственно программистов, то есть на конструировании программных продуктов.

В центре процесса разработки по версии XP стоит принцип ежедневного тестирования, которое должно проводиться командой из двух программистов. Таким образом, ситуация, при которой каждый разработчик изолируется в своей кубышке, а создаваемый им впервые код тестируется специализированной группой лишь через месяц, решительно исключается. Интеграция модулей кода производится сразу после разработки, а тестирование — незамедлительно после интеграции. Практика запуска всех существующих контрольных сценариев на каждом этапе интеграции кода позволяет убедиться в том, что изменение любого отдельно взятого модуля не приводит к деградации остальных. Таким образом, метод XP

---

[1] Термин «жучок» (bug), очевидно, появился еще в эпоху наиболее древних (до 1950-х годов) компьютеров, в которых реле выходили из строя вследствие попадания в них всякого рода насекомых.

[2] Kent Beck, Extreme Programming Explained (Reading, MA: Addison-Wesley, 2000), p. xvi.

выделяется сжатыми циклами выпуска и еще более быстротечными итерациями проектирования.

Метод экстремального программирования ориентирован на проекты, допускающие конструирование силами компактных групп (численностью от двух до десяти разработчиков), участники которых напрямую получают информацию от заказчиков. Процесс XP, таким образом, управляется самой группой разработчиков, а центральное место в методологии занимает программист. Коммерческие специалисты фактически отстраняются от руководства, ограничиваясь «поощрительными» функциями. Управленческая инициатива в группе принадлежит инструктору — программисту, ответственному за процесс разработки в целом.

Буква «X» в аббревиатуре XP означает «экстремальный». Эту характеристику многие деятели нашей индустрии признали вполне удачной. Впрочем, есть и такие, которые, уважая суть концепции, с некоторым презрением относятся к ее названию[1]. По-моему, в публикациях, посвященных этой «новой» методологии, равно как и в практической деятельности по реализации ее принципов, очень много полезного. Полагаю также, что материалы сайта http://www.extremeprogramming.org помогут вам сориентироваться в том, какие методы этого заметного (пожалуй, даже крикливого) течения стоит принять на вооружение. Хотя, если уж следовать духу XP, получается, что вычленять из этой методологии отдельные элементы или смешивать ее с другими стилями нельзя — либо вы «экстремал», либо нет. Меня на это не купишь, хотя мотивация мне вполне понятна. Поборники экстремального программирования пытаются сказать примерно следующее: «Либо дисциплинируйтесь, либо убирайтесь из профессии — вам в ней нечего делать!» Вот это мне по душе — надеюсь, и вам тоже.

Гибкая разработка

В поисках новых методов для себя и своей команды вам еще не раз предстоит столкнуться с термином «гибкий» (agile). Любой эффективный метод разработки программного обеспечения является таким по свой природе. Гибкость означает способность к адаптации, к изменяющимся требованиям и развивающейся технологии, в том числе на этапе конструирования проекта. Мнения большинства специалистов в области разработки программных средств сходятся в одном: гибкость — это священный Грааль всех методик разработки. Существует даже общественная организация, состоящая из ведущих специалистов по разработке программных продуктов и ставящая своей целью пропаганду концепции гибкости[2]. Эти деятели даже опубликовали манифест, в котором обозначены четыре основные проблемы, свидетельствующие, в зависимости от решения, о гибкости тех или иных методов[3].

1. Отдельные лица и взаимодействие или процесс и инструментальные средства.
2. Функционирующее программное обеспечение или комплексная документация.

---

[1] См. колонку «Feedback» в журнале «Software Development» за ноябрь 2001 года.
[2] См. http://www.AgileAlliance.org.
[3] См. Alistair Cockburn, Agile Software Development (New York: Addison-Wesley, 2002), Appendix A.

3. Сотрудничество заказчиков или переговоры по контракту.
4. Реакция на изменения или следование плану.

Признавая определенную ценность положений, расположенных в правой части списка, авторы придают большую значимость его левой части.

Между так называемой «гибкой» школой и экстремальным программированием много параллелей. Действительно, в совещании, на котором был принят манифест гибкости, участвовали, помимо прочих, основатели концепции XP. На самом деле методология гибкости — это, скорее, совокупность принципов, применимая к любым процессам и мероприятиям в области планирования, направленным на конструирование программных средств. В главе 6, рассуждая о методах технического проектирования, я упомянул термин[1] «адаптивная разработка программных средств», имея в виду деятельность, обусловливающую ваш успех в роли технического лидера. Так вот, адаптивная разработка тоже происходит от гибкости.

Мнения большинства специалистов в области разработки программных средств сходятся в одном: гибкость — это священный Грааль всех методик разработки.

В отличие от других методов, рассматривающих предвидение изменений как малозначащий фактор, на который не следует обращать внимания, или как тенденцию в процессе разработки, которой нужно сопротивляться, концепция гибкости призывает к конструктивному и адаптивному реагированию на изменения. В этом свете XP трактуется как подготовительная стадия к внедрению философии гибкости. Любой метод, предусматривающий фиксацию требований и следование предопределенному процессу, приводит к созданию не вполне адекватного программного продукта. Приведу цитату из Кокберна (Cockburn).

> «Некоторые считают, что необходимым условием успешности проекта является следование заданному процессу. Те, кто придерживается такого мнения, тратят уйму энергии на проверку его соблюдения. Человек, твердо убежденный в том, что именно в процессе вся суть, может очень долго не обращать внимания на несоответствие между практикой следования процессу и его исходом»[2].

Иначе говоря, методология гибкости снимает с нас шоры и открывает глаза на те вещи, которые рабы процесса не замечают.

Перечисление этапов процесса разработки, основывающегося на методологии гибкости, противоречит самому ее духу. Методология гибкости рассматривает процесс разработки как опыт сотрудничества и взаимодействия между программистом и заказчиком. Эта позиция, иногда трактуемая исключительно как методологическая тенденция, как мне кажется, являет собой единственный способ обуздать трудный процесс создания качественных программных средств.

---

[1] Этот термин взят из одноименной книги Джима Хайсмита (Jim Highsmith). См. библиографию.
[2] Cockburn, op. cit., p. 5.

Я крайне рекомендую вам порыться в литературе, посвященной гибким методам. Ресурсов по этому вопросу великое множество[1]. Не менее важно оценивать по стандарту гибкости те методы, которыми вы пользуетесь в данный момент. Представьте себе ситуацию, в которой плохо сформулированное коммерческое требование уточняется ближе к сроку завершения кодирования. Сможет ли ваша группа отреагировать на такое изменение без недопустимых задержек? Когда требования меняются во время цикла разработки, большинство из нас проклинают все вокруг. Тем самым мы пытаемся дистанцироваться от реального положения вещей — ведь чем глубже мы разрабатываем проблему реализации требования, тем больше двусмысленности обнаруживаем, а значит, возвращаемся к этапу определения продукта. Гибкие методы культивируют определенное восприятие неопределенности.

Мастерство — ядро любого успеха

Приведенный обзор методологий разработки приводит, как мне кажется, к единственному выводу — разрабатывать программы должны не инженеры, а мастера. Понятие мастерства не всегда легко поддается осмыслению. Я пришел из академической науки и на раннем этапе своей деятельности занимался инженерией; по этой причине я сперва отказывался признавать приоритет искусства перед наукой в процессе создания программных средств. Впрочем, вскоре я понял, что «сопротивление бессмысленно»[2]. Посмотрим, под каким заголовком вышел один из ведущих технических журналов в 1990 году[3]:

> «Производители программных средств страдают от недостаточного контроля за качеством. По мере того как программные продукты становятся все более сложными, компаниям становится все сложнее контролировать миллионы строк кода сложных пакетов»[4].

В статье под этим заголовком речь идет о зрелости программных продуктов (см. главу 6, в которой упоминается модель оценки зрелости продуктов) и количестве ошибок в расчете на строку кода. Там обсуждаются методы измерения количества ошибок, но совершенно не уделяется внимание наиболее важной проблеме: программные продукты создают люди, а отнюдь не процессы, исполняемые автоматами.

Программные продукты создают люди, а отнюдь не процессы, исполняемые автоматами.

Кто создает программные продукты? Мастера. Осознав, что мастера совершенно не пренебрегают наукой и инженерией, вам будет легче признать, что искусство (мастерство) главенствует над научным подходом. Рекомендую поразмыс-

---

[1] Примеры приводятся в работе Cockburn, op. cit. Обзор имеющихся ресурсов имеется также на сайте http://www.adaptivesd.com.

[2] Похоже на «Звездные войны», правда?

[3] Как вы помните, тогда не было ни Windows, ни графических пользовательских интерфейсов.

[4] Electronic Business, October 15, 1990, p. 147.

лить над тем, как нам всем прийти к достойному балансу искусства и науки/инженерии в себе, чтобы обеспечить превращения кода в полезные и качественные программные продукты. Я абсолютно согласен с Питом Макбрином (Pete McBreen) в том, что он пишет в предисловии к своей книге по мастерству разработки программных продуктов:

> «Мастерство знаменует собой возвращение к корням разработки программных средств. Хорошие разработчики понимают (и всегда понимали), что программирование — это деятельность из области искусства. Какими бы потаенными и подробными техническими знаниями ни обладал разработчик, в конечном итоге процесс разработки приложений сводится к ощущению и опыту. Можно знать самые изощренные детали языка программирования Java, но при этом, не чувствуя эстетической стороны программ, так и не стать достойным разработчиком. Если же человек чувствует процесс разработки программных средств, знание конкретных технических деталей уходит далеко на второй план. Лучшие разработчики постоянно учатся новым технологиям и методологиям; постижение очередной технологии для разработчика должно стать обыденным занятием»[1].

Следуя духу мастерства, создайте собственную методологию разработки. Учитесь у коллег, сумевших документировать изобретенные ими процессы, и пусть опыт станет самым верным средством проверки методов[2]. В конце концов, суть науки сводится к постижению истины путем экспериментов. Таким образом, принимаясь за работу с позиции мастера — с тем, чтобы создать достойное программное обеспечение, — вы действуете в полном соответствии с научным и инженерным принципом.

Следуя духу мастерства, создайте свою собственную методологию разработки. Учитесь у коллег, сумевших документировать изобретенные ими процессы, и пусть опыт станет самым верным средством проверки методов.

Технологические революции

«Мыльные пузыри» в рядах программных продуктов регулярно создают излишний шум. Это явление я бы назвал «интеллектуальным зомбированием». Проекты, о которых шли беспрерывные разговоры, не приведшие, однако, к фактической реализации, весьма многочисленны. Упомяну лишь некоторые несбывшиеся обещания и беспочвенные заявления[3].

- Подключенный к сети компьютер за 500 долларов избавит мир от Windows.
- MSN уничтожит AOL (не так быстро — AOL нынче владеет Time-Warner).
- Intuit, несмотря на противостояние с Microsoft, удалось создать более удачный продукт — Quicken.

---

[1] McBreen, op. cit., p. xv.

[2] Такой подход рекомендовал сам Леонардо (см. главу 6).

[3] Этот список я по большей части позаимствовал у Дэвида Корси (David Coursey) — редактора рубрики AnchorDesk на портале http://www.zdnet.com (конкретнее — из статьи, опубликованной им 12 октября 2001 года).

- OS/2 — это DOS лучше «настоящей» DOS и Windows лучше «настоящей» Windows.
- Разработчики настольных прикладных систем дружно перейдут на Java. (Ну это мы еще посмотрим!)

Прецедентов великое множество, и, конечно, здесь они отражены далеко не полностью. Этим своим списком я лишь хочу напомнить о том, что, сталкиваясь с очередным Сенсационным Откровением компьютерной индустрии, стоит быть осторожным. «Технологии нового поколения» — выражение, употребляемое во многих рекламных кампаниях — как правило, действительно дорабатываются лишь с приходом очередного поколения.

Революции в нашей отрасли, бесспорно, случаются. Впрочем, симптомы грядущих изменений, как правило, прослеживаются задолго до окончательного утверждения новых идей и их воздействия на процесс разработки. Вы должны научиться предсказывать тенденции, обещающие заявить о себе через некоторое время. В «Искусстве войны» Сун Цзу (Sun Tzu) писал:

> «Не разбивайте лагерь на сложной местности. Объединяйтесь с союзниками на пересечении крупных путей. Не задерживайтесь на изолированных позициях. Находясь в окружении противников, старайтесь обмануть их. Деритесь только при крайней необходимости»[1].

Следуя этим мудрым советам, с тем чтобы избежать «крайней необходимости», разрабатывайте техническую стратегию революции задолго до наступления самой революции. Сун Цзу говорит о «сложной местности» — полагаю, эта ситуация применима и в разработке программных средств. Домысливая китайского философа, я рекомендую вам в ходе долгосрочного планирования учитывать следующие стратегические проблемы.

- *Не разбивайте лагерь*. Знания ваших подчиненных программистов не должны ограничиваться единственным языком и одной инфраструктурной реализацией архитектурных решений коммерческих задач.
- *Объединяйте усилия*. См. предыдущий пункт. Обращайтесь к зарекомендовавшим себя решениям от разных поставщиков — чем больше их будет, тем лучше. Время от времени привлекайте консультантов — пусть делятся с сотрудниками новыми знаниями. Стимулируйте процесс обучения за счет построения силами сотрудников компактных макетов, нацеленных на проверку применимости новых методик к разработке более крупных проектов.
- *Не задерживайтесь*. Успехи компании иногда порождают в ее сотрудниках инертность, которая не позволяет им внимательно отслеживать новые ходы конкурентов и находить возможности для дальнейшего роста. Вы должны стараться сохранять гибкость по части применения новых методов и инструментов. Не отвыкайте от неудобств — даже в том случае, если в данный момент вы их не ощущаете.

Как отслеживать новейшие тенденции? Для этого существует литература, которую нужно читать в больших количествах, и Интернет, пользоваться которым нужно с умом. Выберите несколько надежных источников информации и обра-

---

[1] Цитирую по: Sun Tzu, The Art of War. Ed. James Clavell (New York: Dell Publishing, 1983), p. 37.

щайтесь к ним, как только вам потребуется что-то узнать[1]. А самое главное — путем плотного взаимодействия с начальником и подчиненными выработайте комплексную стратегическую программу, то есть руководство к действию в постоянно меняющемся мире, в котором мы, разработчики, все пребываем.

Экономические несчастья

Слова «экономические» и «несчастья» часто употребляются вместе. Не так уж часто можно услышать об экономических радостях, так? Из последних событий в контексте экономических несчастий в памяти всплывают лопнувшие пузыри интернет-компаний. Как бы там ни было, риск — это неотъемлемый элемент будничной деятельности специалистов в области разработки программных средств, и учиться на неудачах мы должны не менее тщательно, чем мы учимся на успешных прецедентах. Крах интернет-компаний, конечно, можно свалить на инвестиционных банкиров, но, пожалуй, для нас как для профессиональных разработчиков такая позиция представляется не слишком продуктивной. Спору нет — жадность, алчность и все прочие присущие капитализму пороки сыграли в крушении этих компаний не последнюю роль. Тем не менее на первый план во время экономического несчастья выходят люди, такие как вы и я, которые теряют работу, испытывая попутно эмоциональные и интеллектуальные страдания. В этом разделе я постараюсь объяснить, какие конструктивные действия вы можете предпринять, чтобы предотвратить наступление кризиса.

Пожалуй, самый важный урок, которому нас учат многочисленные кризисы, состоит в следующем: если компания перестает приносить доход, значит, вероятно, конец близок[2]. Таким образом, лучший способ предотвратить наступление кризиса в компании — это следить за издержками разработки и стремиться к рентабельности. Признаю: совет не слишком оригинален — это все равно что сказать: «Если хотите заработать на Уолл-стрит, покупайте задешево, а продавайте втридорога». Совет на самом деле правильный, но лишь при условии верно выбранного времени. Таким образом, очевидно, что во главе угла находится именно время.

И вот это самое обстоятельство подводит меня к основной мысли в экономическом контексте. Если правильный выбор времени играет такую важную роль, значит, синхронизировать экономические часы вашей компании с реальной ситуацией можно лишь за счет последовательности. Что я имею в виду? Занимая лидирующую позицию в своей команде, вы должны придерживаться нижеследующих правил.

- Вы — не только технарь, но и бизнесмен. О стоимости, таким образом, нужно думать не меньше, чем о коде.
- Составьте представление о реальных затратах на содержание вашего отдела. Ежемесячно собирайте данные о стоимости технических средств, оборудования, услуг, инструментов и заработной плате персонала.

---

[1] Как мне кажется, для начала подойдет сайт TechRepublic (http://www.techrepublic.com). На нем публикуются вполне добротные тематические обзоры.

[2] Интернет-компании славились (и славятся до сих пор) бездоходностью. Amazon.com вот уже несколько лет приносит прибыль, и, вероятно, эта компания — скорее исключение из общего правила; впрочем, время покажет. Лично мне будет жаль, если они обанкротятся. В конце концов, неужели многих сотен долларов, которые я перечислил им за годы нашего знакомства, недостаточно для выживания?!

- Сравнивайте окончательную фактическую стоимость проекта с предварительными расчетами. Оценивайте расхождения между сметой и расходами и на основе этого опыта учитесь подходить к составлению смет с более реалистических позиций. Не забывайте поговорку: «День профилактики стоит года лечения».
- Держите зарплаты сотрудников под контролем. Платите им столько, сколько они заслуживают, сверяясь при этом с рыночной конъюнктурой[1].

Увольняя сотрудника, позаботьтесь о его дальнейшем трудоустройстве — пишите рекомендательные письма[2] и звоните работодателям. Почувствовав, что скоро могут уволить вас самого, немедля приступайте к поиску нового места работы (если, конечно, это возможно) — не ждите, пока извещение об увольнении придет по факсу. Перейти с одной работы на другую значительно проще, чем выбраться наверх из армии безработных.

Одиночество руководителей

Лидеры, осваивающие во главе своих групп новые территории, часто ощущают свое одиночество. Это — цена лидерства. Переход из программистов в разряд руководителей и соответствующее изменение обязанностей не всегда проходят бесследно, и далеко не всем удается избежать депрессии. Как я уже говорил, ждать, что кто-то похлопает вас по плечу и скажет, что вы все правильно делаете, не стоит — этого может и не случиться. Если процесс адаптации к административным функциям еще не закончен, вполне возможно, вы чувствуете себя не слишком комфортно — в эмоциональном плане.

Все, что я скажу по этому поводу, можно, конечно, воспринимать как опыт психологического анализа, но ведь я сам прошел через этот этап — так что, скорее, это воспоминания. Чувство пребывания не в своей тарелке пройдет, поверьте, и есть средства, с помощью которых процесс психологического восстановления можно ускорить.

Уделяйте время исследовательской работе

Как смягчить последствия отхода от привычной деятельности — кодирования? Рекомендую как можно больше времени уделять исследованию новых технологий, написанию кода с помощью не испробованных ранее инструментов и методик. Не стоит, впрочем, заниматься кодированием в рамках текущих проектов — если, конечно, вы не испытываете недостатка в персонале. Ответственность за процесс разработки в целом усложняет наблюдение за промежуточными результатами сотрудников. Поэтому если за счет руководящих обязанностей вы окажетесь единственным разработчиком, не успевшим доделать свою часть кода, ситуация лишь усложнится. Я понимаю, что вы готовы взяться за любую возможность

---

[1] Я уже высказывался на эту тему в главах 1 (см. подраздел «Мотивирование деньгами» в разделе «Слава, почет и деньги») и 3 (см. подраздел «Денежное поощрение и продвижение сотрудников по службе» в разделе «Как сформировать команду и как ее поддерживать»), но здесь имеет смысл повториться.

[2] В некоторых случаях от такой помощи приходится отказываться по правовым причинам. По вопросам корпоративной политики в этой области вам стоит проконсультироваться у начальника.

поработать с кодом, и эта деятельность тоже входит в обязанности лидера, но прошу вас: будьте в этом отношении благоразумны.

Одна из наиболее существенных технических задач лидера сводится к оценке применимости для внедрения в проект новых программных средств, методик и вариантов архитектуры. Вы — лидер и наставник — не сможете обучать сотрудников, не зная реального положения вещей в тех или иных областях. Подобного рода анализ — занятие трудоемкое и требующее серьезной технической подготовки, а это значит, что ваш опыт в роли высококвалифицированного программиста будет весьма кстати.

Как превратить административные функции в инженерные

Как вы знаете, в оценке программной инженерии как методологии разработки я не слишком оптимистичен; тем не менее мне кажется, что инженерные методы вполне применимы к административной деятельности. Автоматизируйте процесс планирования проекта и еженедельного распределения рабочих заданий. В главе 4 я уже касался этой темы, но, по-моему, она заслуживает повторного обзора. Администрирование отнимает много времени, и для всех сфер деятельности руководителя этого времени не хватает; упорядочить поток информации помогут ваши навыки программирования.

Изучите биографии и принципы деятельности руководителей, которым удалось выстроить свои административные обязанности и реализовать амбиции. Помимо составления памяток, телефонных звонков и прочих будничных административных операций, они приложили усилия к процветанию своих компаний, самосовершенствованию и повышению квалификации персонала. Если вы поставите своей целью административную деятельность и не будете всеми силами стараться избегать ее, то сможете достичь сопоставимых высот.

Стратегическое планирование как наука

В предыдущей главе (см. главу 9) я рассказывал о том, как помочь начальству достичь успеха. Может статься, что вашей основной обязанностью окажется стратегическое планирование развития компании в техническом аспекте. Таким образом, вы должны уделить особое внимание совершенствованию навыков создания стратегических планов, в которых должны не просто прогнозироваться возможные векторы развития, а буквально строиться будущее. Это не так-то просто. Многие так называемые «белые книги» — программные документы корпораций — зачастую несут единственную функцию: развеять опасения о непродуманности будущего развития. Впрочем, думать о будущем и планировать его — это несколько разные вещи.

Разрабатывая стратегию на будущее, необходимо изучать прошлое. Проанализируйте старые планы, подумайте, почему некоторые из них увенчались успехом, в то время как другие провалились. Таким образом, вам предстоит стать историком применявшихся в компании технологий и предсказателем тенденций на многие годы вперед. Говорят, «те, кто отказываются учить уроки истории, обречены на ее повторение», — быть может, это и банально, но смысл очевиден.

Планы нужно строить исходя из зарекомендовавших себя моделей — обращать излишнее внимание на «мыльные пузыри» не стоит. Лишь при этом условии стратегическое планирование сможет претендовать на научное наполнение. Кто сказал, что стратегические документы не нужно подкреплять примерами кода? Жизнеспособность новых идей и методик в исследованиях надо наглядно иллюстрировать разделами, посвященными их практической проверке. Конечно, блок-схемы с прототипами архитектурных реализаций очень важны, но, подтвержденные примерами из «текущей деятельности», они приобретают еще больший вес.

Учитесь ценить человеческие отношения

Не сомневаюсь в том, что значительную часть межличностных отношений вы оцениваете вполне адекватно. Но, черт побери, мы же технари, а значит, наша способность к контактам довольно ограничена. Если это не про вас, примите мои извинения. И все же — существует ли «предельная ценность» применительно к межличностным отношениям на работе? С «параноидальной» точки зрения, наверное, да. Впрочем, это совсем не та паранойя, к которой нужно стремиться[1]. Я совсем не имею в виду «слезливую плаксивость» — может статься, вы и в этом меня заподозрили. На самом деле речь о личностной ценности каждого конкретного сотрудника группы, на которого вы затрачиваете время и энергию, пытаясь стимулировать его рост как программиста и потенциального лидера. Жалко, что в Вооруженных силах нынче не в ходу лозунг «будь тем, кем ты способен быть», — ведь лидеры иногда ощущают себя сержантами, муштрующими своих солдат.

Жалко, что в Вооруженных силах нынче не в ходу лозунг «будь тем, кем ты способен быть», — ведь лидеры иногда ощущают себя сержантами, муштрующими своих солдат.

Никогда не забывайте о том, что настоящий лидер пытается подвигнуть своих подчиненных на максимально продуктивную деятельность. Чем больше вы в них вкладываете, тем лучше результат. Следовательно, все случаи формального и неформального общения с сотрудниками, по большому счету, воспринимаются как рабочие. Вы — начальник, и именно в этой роли вас видят и оценивают подчиненные. «Своим парнем», как когда-то в бытность программирования в команде, вам уже не быть. Таким образом, любые отношения с сотрудниками следует выстраивать с учетом вашего положения в компании.

Финал

Глава эта, как известно, называется «Слова без песни». Вот и здорово, что заключительный ее раздел тематически совпадает с названием. Гоняя вас по темам самого разного содержания, я преследовал единственную цель — показать, что в процес-

[1] См. главу 8 — в ней упоминается конструктивная трактовка слова «паранойя», приписываемая Энди Гроуву.

се выпаса котов побеждать хаос можно по-разному. Итак, мы пришли к следующим умозаключениям.

- В контексте распределенных рабочих групп усиливается значимость планирования и проверки проекта.
- Для того чтобы управлять сотрудниками, относящимися к разным культурам, вы должны стать гражданином мира и подавить те национальные характеристики, которые могли бы стать препятствием к взаимопониманию и снизить мотивацию вашего многонационального персонала к успешному ведению деятельности.
- Вам предстоит постоянно совершенствовать применяемые методологии разработки. Одним из необходимых условий успеха является стремление к гибкости всех процессов.
- Технологические революции поддаются прогнозированию, а значит, к ним можно подготовиться. Стратегическое планирование на основе понимания предстоящих перемен — вот средство, позволяющее предотвратить излишнюю суету в случае их наступления.
- Экономические кризисы случаются регулярно, и ничего с этим не поделаешь. Стремлением к прибыльности своей компании вы должны стараться свести к минимуму последствия неблагоприятной экономической ситуации. Для начала научитесь руководить отделом не только как технарь, но и как бизнесмен.
- Лидерство иногда приводит к ощущению одиночества. Существуют, впрочем, средства, позволяющие безболезненно выйти из этого состояния. Откажитесь от самоизоляции. Культивируйте в себе активный творческий подход к исследовательской деятельности, администрированию, планированию и общению с людьми.

Лидерство — это увлекательное приключение. Я уверен, возможностями для исследования и творчества наша индустрия изобилует. Мы, люди технической направленности, работая в ней, имеем все шансы расширить свой кругозор и стать пионерами кадрового и программного обеспечения. Итак, цените каждый миг рабочего дня и извлекайте из него пользу — как для себя, так и для подчиненных.

Послесловие

Снова в плавание...

Так, ладно — пора вам закругляться и возвращаться к повседневным обязанностям. Послесловие, как видите, тоже метафорично — действительно, я хочу, чтобы вы трактовали свои поступки на руководящем посту как плавание корабля в неспокойном море. Морем считайте индустрию разработки программных продуктов в том виде, в котором она существует сейчас — полагаю, в XXI веке она покажет себя не менее динамичной, чем в веке XX. Чтобы без особых трудностей проплыть через это море, вам потребуются надежный руль для поддержания курса, большой и прочный парус, чтобы ловить ветер, и якорь для устойчивости. Надеюсь, моя книга поможет удержать вашу шлюпку на плаву.

Руль

Кораблем управляют с помощью руля. Чем управляется ваша профессиональная деятельность? В идеале она должна основываться на двух важнейших императивах: «*сконцентрируйся* и *лидируй*!» По мере совершенствования лидерских качеств и преследуя цель производства отличных программных продуктов, не забывайте о том, что повышать концентрацию нужно постоянно. Ищите любую возможность попрактиковаться в деле лидерства — если не закрывать глаза на окружающий мир, такие случаи будут подворачиваться на каждом шагу. Все принципы и методики, рассмотренные на предшествующих страницах, направлены на достижение одной-единственной цели: помочь вам стать лидером и совершенствоваться в этом качестве. Чтобы выделить время на лидерскую деятельность, нужно предварительно достичь определенных высот на ниве руководства — таким образом, сконцентрировавшись на неотложных и наиболее значительных приоритетах, вы должны проявить себя хорошим организатором. Именно в этом заключается смысл «концентрации» — умение не отвлекаться на внешние раздражители помогает уделять максимум внимания решению самых важных задач.

Сфера вашей деятельности весьма обширна — вам предстоит заниматься разными делами, причем некоторые из них очень интересны. Раздражители, помимо прочего, появляются из-за неверного выбора технологий для решения коммерческих задач. Вы должны научиться отметать не подходящие для достижения ва-

шей компанией успеха решения (пусть даже они, по идее, очень хороши) в пользу действительно необходимых вариантов. Подобная систематизация достигается путем планирования и реализации плана. Устранению раздражителей помогает также правильная организация административной деятельности. Внешне прибранный рабочий стол свидетельствует о структурированности мысли, а беспорядок, напротив, выражает неопределенность по поводу приоритетов. Обратите внимание: я сказал «внешне». На самом деле, важнее всего ваше внутреннее отношение к рабочим вопросам.

Вступив в бой с процессом разработки, вы будете вынуждены пересмотреть свою систему приоритетов. Стоят ли люди на первом плане в вашей иерархии ценностей? Они должны быть именно там — в конце концов, никто кроме людей не напишет для вас код. Чем внимательнее вы будете относиться к своим сотрудникам, тем больше вероятность, что они станут воспринимать ваши задачи как свои и делать все для их решения. Многие считают так: «я работаю на компанию такую-то». Этого мало — нужно, чтобы они говорили: «я работаю на _____ (впишите сюда ваше имя)». Преданными люди становятся не сразу — для того чтобы сформировать в них это качество, необходимо постоянно терпеливо принимать участие в их рабочей жизни.

Парус

Ветер надувает паруса, заставляя корабль двигаться. Атмосфера в индустрии разработки программных средств отличается динамичностью, поэтому «ветров» в ней бушует предостаточно. Способность идти (в зависимости от ситуации) по ветру или против ветра есть необходимое условие продвижения в верном (в контексте целей вашей компании) направлении. От того, к какому «лагерю» вы принадлежали в бытность занятий программированием, ничего не зависит — имейте в виду, в последние десять лет бури дули часто и сильно.

Если вы привыкли ориентироваться на продукцию Microsoft, значит, ветры больше всего дули с северо-запада. Направление ветров за последние несколько лет можно определить — все они явились результатом тщательного (и не очень) планирования. В чем, по-вашему, смысл Windows 98 Second Edition? А как насчет бесчисленных служебных пакетов для Visual Studio? Сможете ли вы навскидку припомнить все номера вышедших в свет версий ADO? Пробовали ли вы запускать утилиту от Microsoft, определяющую наиболее подходящую для конкретной машины версию ADO[1]? Не успели вы привыкнуть к Windows 2000, как на рынке появляется Windows XP (более дорогая) — так ведь? Каждая последующая версия Visual Studio знаменует для программистов VB смещение парадигмы и обещает кардинальным образом изменить методы написания кода.

Изменения не обошли стороной сообщества Java и C++, и, скорее всего, ими дело не ограничится. Компания Sun несколько лет держала название своего нового языка в секрете — собственно, оно шло вразрез с традицией именования языков. Первоначально язык предполагали назвать «Oak», но, как оказалось, это имя уже

[1] Я имею в виду программу ComCheck, которая распространяется Microsoft бесплатно и призвана помочь разобраться в создаваемом ADO библиотечном кошмаре.

занято. Судя по всему, имя «Java» придумали с расчетом на то, чтобы «встряхнуть» быстро развивающийся рынок разработки интернет-проектов. И, действительно, все мы почувствовали встряску. Кроме того, есть пространство CORBA и COM. Или, возможно, CORBA и COM+? Или CORBA и SOAP? Кто знает — может, сформировавшуюся брешь закроет XML? Ну, в общем, вы меня поняли...

В условиях постоянного ужесточения бизнес-требований и развития технологий изменения неминуемы. Пользователи едва успевают за этой динамикой, не говоря уже о программистах (ваших, моих, пашущих на Microsoft и проч.), которые часто повторяют собственные ошибки. Все эти условия благоприятствуют сильным ветрам.

Для того чтобы устоять в бурю, нужно постоянно стремиться к созданию качественных программных продуктов и регулярно повышать планку этого самого «качества». Вам придется утрясать сроки с начальством, препираться со специалистами по анализу рынка насчет задач, на решение которых способен тот или иной продукт, как-то разбираться с недостатками сетевой инфраструктуры... Этот список можно продолжать бесконечно. Ничего удивительного в том, что продукты нашей деятельности зачастую несут оттенок незаконченности. Дело даже не во впечатлении — они действительно являются таковыми. Вероятно, лишь в идеальном мире можно было бы создать условия для производства абсолютно безошибочного продукта. Впрочем, на пути к идеалу мы движемся слишком медленно — непозволительно медленно. Сосредоточьтесь на качестве, и вы сможете завоевать уважение коллег и успех на рынке.

Для того чтобы постоянно поднимать планку «качества», нужно быть увлеченным своей работой — только при этом условии, кстати говоря, ваши лидерские начинания имеют шанс на развитие. Именно увлеченность позволяет удерживаться на плаву в шторм. Увлеченность формируется за счет баланса рабочей деятельности, с одной стороны, и приятных для вас мелочей жизни, с другой. Невоздержанность в той или иной сфере жизни приводит к потере увлеченности, а следовательно, к усталости и краху. Старайтесь удержать увлеченность работой и любовь к жизни — превращайте все свои начинания в увлекательные приключения.

Еще один момент касательно «баланса». В современных рабочих условиях баланс предстает в виде «смешения». Это утверждение особенно справедливо по отношению к удаленным сотрудникам — вне зависимости от того, работают они полный рабочий день или садятся за кодирование вечерами. При наличии карманных компьютеров, ноутбуков, постоянных интернет-соединений через виртуальные частные сети и прочих инструментов нашей деятельности появляется возможность не покидать рабочее место. «Балансирование» в таких условиях предполагает проведение 15 часов кряду в одном и том же месте — там, где можно заниматься и работой, и личными делами. Как организовать для себя отдых — необходимое условие «подзарядки» мозгов? Ничего определенного на этот счет я сказать не могу, но отдыхать необходимо — иначе вы рискуете потерять увлеченность и, в конечном итоге, «сгореть». Если топлива не осталось, зажечь огонь заново не так-то просто.

Якорь

Якорь придает нам устойчивость, и время от времени мы к нему прибегаем. Одно лишь наличие якоря на борту позволяет чувствовать себя спокойно даже в штормовую погоду. Роль якоря в нашей профессии исполняют лидерские навыки. Выпас котов — деятельность, которой я с помощью этой книги пытаюсь вас научить, — предполагает проявление лидерских качеств на уровне «выше среднего программиста». Программист в силу особенностей своей профессии сориентирован на объекты, которыми можно управлять. Создавая объект с определенным количеством открытых интерфейсов, мы ожидаем, что при обращении из другого объекта проявят себя только они. Люди устроены немного по-другому. За некоторый период времени они могут несколько раз сменить маску, а в экстренных ситуациях — измениться до неузнаваемости. Ваша задача — научиться работать с людьми в их самых необычных обличьях, с тем чтобы выстроить их однонаправленную деятельность. Относительно направления, в котором всем предстоит двигаться, вы не должны испытывать никаких сомнений. Кроме того, вам нужны *факторы притяжения* (attractors). Я в данном случае не имею в виду хитрые конструктивы из теории хаоса. Я о более простых вещах — необходимо научиться привлекать к себе людей за счет уверенного лидерства. Даже самый талантливый руководитель на это не способен. Творческому и продуктивному программисту это тоже не под силу. Но, сосредоточившись на развитии в своем характере лидерских качеств, *вы* сможете добиться в этой области нужного результата. Дело не в методиках. Совершенствовать методики позитивного мышления можно сколько угодно — привлекательными для сотрудников они все равно не станут. Незаменимым в этом отношении оказывается великолепие последовательного и продуманного лидерства.

Вам предстоит проводить регулярный анализ эффективности своего лидерского поведения. Рассмотрим аналогию. Как известно, программные продукты иногда перестают работать из-за конфликта версий библиотек DLL. Это явление, которое время от времени ставит под угрозу результаты труда разработчиков, называют «библиотечным кошмаром» (DLL Hell). Правда, к счастью, все подобные проблемы можно решить разом — просто переустановив операционную систему. Примерно этим вам предстоит время от времени заниматься в контексте своих лидерских качеств. Нельзя беспрерывно накапливать руководящие методики и надеяться, что таким образом все проблемы решатся сами собой. Иногда полезно начинать отсчет «с нуля» — каждый божий день стараться мыслить вне сложившихся стереотипов. «Стереотипом» в данном случае представляется существующий метод ведения дел в отделе. Переосмысливать фундаментальные принципы лидерства нужно по мере необходимости. Что сделать из того, до чего вы еще не додумались? Какие практики следует прекратить? Трудно надеяться на то, что стать лидером программистов вам удастся лишь по той причине, что вы занимаете пост менеджера, руководителя группы разработчиков, начальника отдела разработки — да хоть менеджера по информации! Право на лидерство нужно еще заслужить — для этого необходимо осознать свои слабые места и стремиться к их устранению.

Надеяться на то, что за ваши достоинства вас будут превозносить до небес, не стоит — но, по крайней мере, они (достоинства) возымеют действие на сотрудников компании. Ваши недостатки, наоборот, не пройдут незамеченными — о них будут говорить, в том числе и вам самим; да, так устроен мир, ничего не поделаешь. В общем, налегайте на свои достоинства.

Кое-какой материал в этой своей книге я представил в форме наставлений. На самом деле их было предостаточно. Может быть, именно по этой причине вы дочитали книгу до конца — вам нужна была проповедь. Надеюсь, она прошла небезрезультатно и поможет вам в последующей деятельности.

Дж. Хэнк Рейнуотер
Январь 2002

Приложение А

Как ухаживать за живностью — электронный администратор

У программ с открытым исходным кодом есть много достоинств и совсем немного недостатков. Как известно, главное для нас — стандарты. В то же время при наличии исходного кода у нас появляется возможность вводить собственные стандарты. Я решил предоставить в ваше распоряжение исходный текст электронного администратора — быть может, с ним вы сможете добиться успеха чуть быстрее, чем без него. Должен предупредить, что эта программа предназначена исключительно для личного пользования — соответственно ее нельзя перепродавать, а также сначала переделывать, а потом перепродавать. Для ее загрузки зайдите на сайт http://www.piter.com.

Электронный администратор (подробнее см. главу 4) написан в среде Visual Basic 6.0 с установленным служебным пакетом 5. В качестве базы данных применяется Access 2000. Помимо стандартных элементов VB, я обращался к следующим средствам:

- интерфейсы источников данных от Microsoft;
- библиотека Microsoft ActiveX Data Objects 2.6;
- Crystal Reports версии 8.5 (отдельные компоненты)[1];
- Microsoft Direct Speech Synthesis;
- Microsoft Agent Control 2.0;
- Microsoft Scripting Runtime;
- Microsoft Direct Speech Recognition.

---

[1] В Crystal Reports 8.5 какое-то невероятное количество библиотек DLL. У меня установлена полная версия этого пакета для разработчиков, и для того чтобы с отчетами можно было хоть что-то делать,требуется великое множество файлов. При этом ни сам пакет, ни мастер развертывания (Deployment Wizard) в VB 6.0 не справляется с идентификацией этих файлов.

Помимо стандартных компонентов, в этом проекте я обратился к довольно древнему элементу управления календарем, созданным MSCAL.OCX[1] из поставки Access 97, и специальным элементом управления датами под названием MyData.OCX, который вы можете скопировать с сайта вместе с исходным кодом.

Можете заменять ссылки и компоненты как вам заблагорассудится. Только не забудьте соответствующим образом скорректировать код.

INI-файл регулирует конфигурацию программы и жестко задает структуру каталогов. Каждому «ресурсу», определенному в таблице базы данных с аналогичным именем, должен соответствовать каталог для экспорта отчетов. В каталоге с шаблонами содержатся задействованные в программе файлы Crystal Reports. Стоит только открыть самораспаковывающийся файл с кодом, и он автоматически создаст нужную структуру каталогов. В тот же момент база данных наполнится шаблонными данными.

В базе данных есть ряд таблиц соответствия, которые вы можете заполнить так, как сочтете нужным. На то, чтобы сделать еще несколько таблиц, мне элементарно не хватило времени — в частности, я не добрался до таблицы проектов, которую можно было бы соединить с таблицей заданий. Это решение, надо сказать, упростило процесс генерации отчетов средствами программы Crystal Reports, а потому имейте в виду: любое изменение скажется на шаблоне отчетов. Если хотите, можете изменить мой код — мне, конечно, будет интересно увидеть результат. Возникнут вопросы — пишите на адрес herdingcats@mindspring.com. Постараюсь на все отвечать.

А теперь — несколько слов о дополнительных экранах программы, которые способны оказать существенную помощь в деле организации личных информационных потоков. На рис. А.1 изображено родительское окно.

Экран Today, показанный на рис. 4.3 в главе 4, несомненно, важен для систематизации административных функций, однако им одним программа не исчерпывается. Согласно моей теории планирования руководящей деятельности, любое задание нужно рассматривать в трех основных измерениях: Project (Проект), Source (Источник) и Assigned (Назначенные задания). Представление Project (рис. А.2) помогает отслеживать ход выполнения всех заданий в рамках проекта. Представление Assigned (рис. А.3) демонстрирует распределение заданий между сотрудниками. Наконец, представление Source (рис. А.4) напоминает о том, кто (например, какой процесс или комитет) требует результата от вас. Все это дочерние окна интерфейса MDI с изменяемым размером, поэтому открывайте их столько, сколько хотите.

По информации с каждого из этих экранов генерируются отчеты, которые можно отправить по почте сотрудникам, принести на собрание, предоставить начальнику, просмотреть на предмет согласования ваших планов с деятельностью других подразделений компании. До тех пор пока приложения для коллективной работы не достигнут определенного уровня зрелости, несогласованность в дей-

---

[1] Этот элемент OCX вызывает у меня некоторые подозрения. Он вроде как не обнаруживает зависимостей от других файлов (помимо стандартных файлов Windows API), однако заставить его работать без Access 97 мне не удалось. Я соорудил собственную версию этой программки на основе элемента управления календарем из поставки VB 6.0, так что если при загрузке кода вы наткнетесь на неработающую ссылку, милости прошу!

ствиях между отделами при отслеживании разных типов заданий будет оставаться обычной практикой — при этом каждый из отделов будет навязывать собственный стиль руководства. Поэтому для управления списками заданий часто привлекается метод «вырезания и вставки», а управленческие реалии, которые больше напоминают страшный сон, некоторые именуют не иначе как «казнь через разрезание на тысячу кусочков».

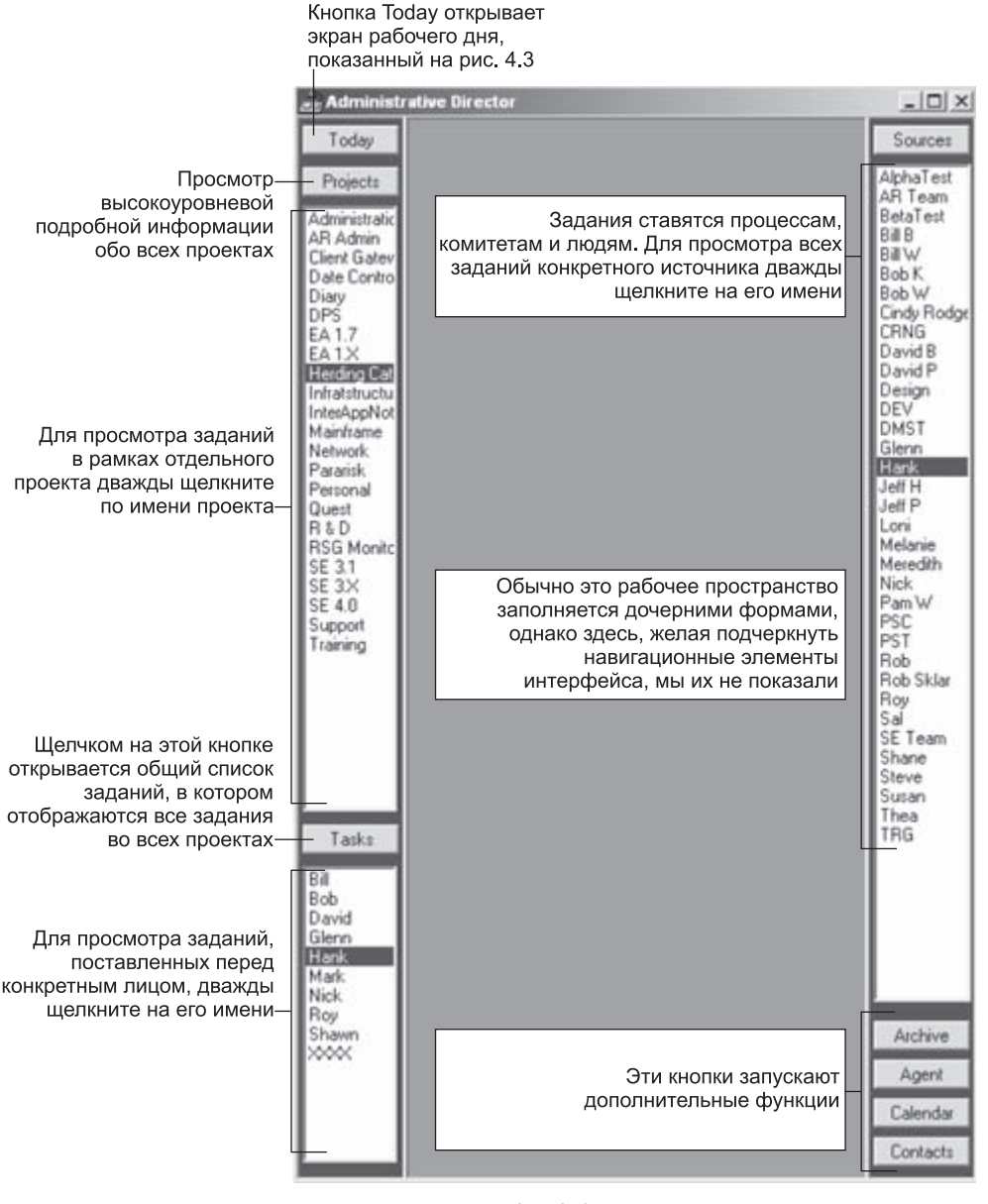


Рис. А.1. Родительское окно (MDI) без дочерних окон

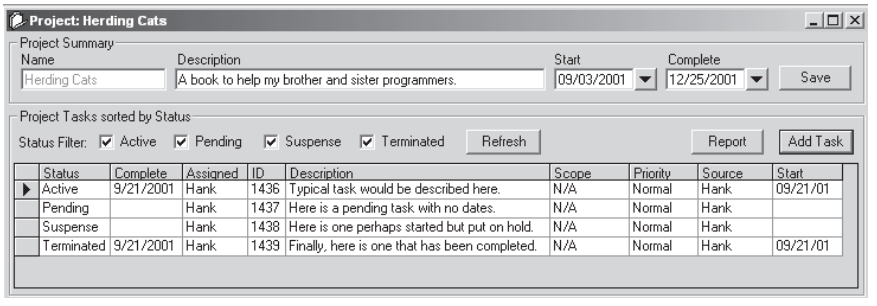


Рис. А.2. Представление Project

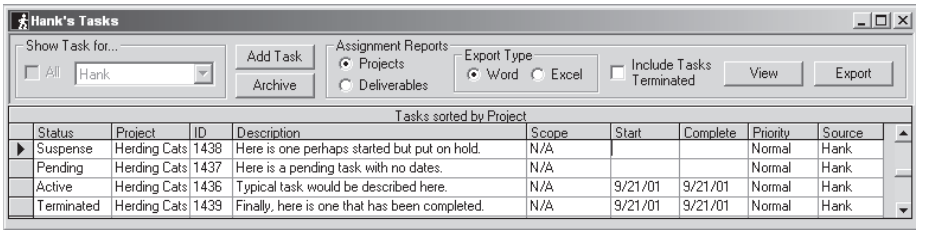


Рис. А.3. Представление Assigned

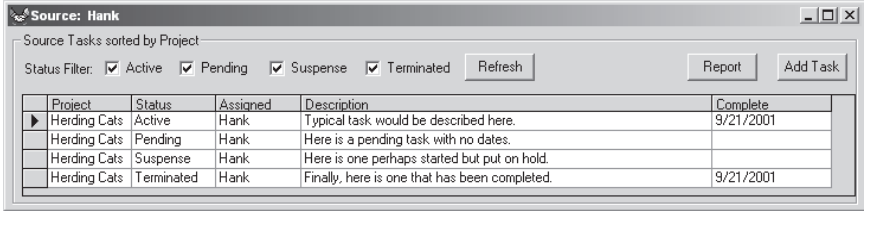


Рис. А.4. Представление Source

Три окна со списками, расположенные по периметру родительского окна (см. рис. А.1), предназначены для быстрого доступа к трем основным представлениям. В этих окнах содержатся списки имен проектов, ресурсов, которыми можно назначить новые задания (в основном это имена людей), и источников заданий, над которыми вы работаете или которые отслеживаете. По двойному щелчку на любом пункте списка на экране появляется новое представление, в котором задания рассортированы в соответствии с назначением представления.

Я предпочитаю выделять отдельный элемент Source для отслеживания тех заданий, которые передо мной ставит моя многоуважаемая руководительница. Забыть предписание начальницы — это же так глупо (не говоря уже о том, что этот опрометчивый поступок ставит под угрозу карьерные перспективы)!

Выполнение функций, представленных в открываемых щелчком правой кнопкой мыши контекстных меню, обеспечивается несколькими объектами-контейнерами. Испробуйте их или хотя бы взгляните на соответствующий код обработки событий.

Ну и все на этом. Хотите узнать больше — поройтесь в исходном тексте. Забавляйтесь с живностью, но не обижайте ее!

Приложение Б

Как дать скотине в глаз — критический обзор кода электронного администратора

В главе 6, в разделе «Кодовая полиция», я объяснял, как стать кодовым полицейским. Сохранять объективность при чистке собственного кода очень сложно, но я постараюсь. Связав в названии этого приложения свой код со скотным двором, я надеялся подсказать вам, что я собираюсь делать. Все, что я хочу, — это чтобы вы разобрались в процессе критического обзора кода и обратили внимание на некоторые моменты, которые играют существенную роль в процессе проверки кода, написанного подотчетной вам группой.

Имея перед глазами исходный код, вам будет легче читать это приложение. Как я уже говорил в приложении А, скачать код можно с сайта http://www.piter.com.

Контекст и происхождение программного продукта

Код электронного администратора я написал вскоре после прекращения работы над неким веб-проектом, призванным облегчить административную волокиту, сопровождающую любой процесс разработки программного обеспечения. Поскольку времени на написание кода, который мне позже предстояло самому же раскритиковать, категорически не хватало, для иллюстрации нескольких архитектурных принципов я привлек материалы упомянутого проекта. Эксперимент этот, принесший мне немало забавных впечатлений, кажется, удался.

Не скажу, что код, о котором идет речь, может по своему качеству соревноваться с коробочными продуктами, но, в общем, он не так уж плох и с моими административными функциями справляется успешно.

Правила игры

Анализировать код я намерен по методике, изложенной в главе 6. Итак, как я уже говорил, хороший код отличается следующими особенностями.

- *Он пишется в соответствии со стандартами программирования, принятыми для конкретного языка.* В таком случае применяемые разными программистами методики конструирования объектов, обусловленные архитектурой, не будут слишком разниться.
- *Внутри объектов соблюдается строгая связность.* Объект — это несколько больше, чем просто группа процедур; он должен выполнять конкретную функцию. Как известно, сердце не дышит, а легкие не качают кровь.
- *Взаимозависимость между объектами по возможности минимизируется.* В большинстве случаев (в отсутствие существенных доводов в его пользу) взаимозависимость не приводит ни к чему хорошему — она лишь усложняет сопровождение. На последующую изоляцию взаимозависимых объектов затрачиваются серьезные финансовые и временные ресурсы.

Я обращу ваше внимание на ряд слабых сторон кода. По большей части они обусловливаются сжатыми временными рамками и тем обстоятельством, что инструмент, к которому этот код относится, я проектировал исключительно под себя. Пока что я ни разу не утверждал, что безгрешен, поэтому мои самоистязания вряд ли кого-нибудь удивят.

Следовал ли я стандартам?

В основном я следовал стандартам — по крайней мере, мне так кажется. VB я пользуюсь, начиная с версии 1.0, и с годами отношения с кодом складывались у меня (наверное, так же как и у вас) по-разному — был и удачный, и провальный опыт. С моей точки зрения, следование стандартам VB выражается в попытках привести объектно-ориентированные понятия в соответствие с этим языком, который, по правде говоря, отнюдь не полностью поддерживает объектно-ориентированную парадигму.

В главе 4 я изложил понятие задачи, или задания, — основного организующего принципа программного обеспечения. Просмотрев мой код, вы найдете в нем объект под именем clsTasks и вспомогательный объект clsTask. В этих двух модулях классов инкапсулированы пользовательское взаимодействие и все данные, обрабатываемые программой в связи с заданиями. Формы frmTask и frmTasks, ответственные за обработку заданий на стороне графического пользовательского интерфейса, являются дочерними объектами объекта clsTasks. Все прочие объекты, например clsToday, при обработке заданий обращаются к локальным экземплярам clsTasks. Эта схема довольно удачно, как мне кажется, иллюстрирует методику многократного использования объектов.

Модульные объявления объекта clsTasks выглядят так:

```
'---Закрытые объекты и события
Private mo_DataService As clsDataService '---данные объекта
Private mo_PickList As clsPickList       '---список отбора для форм
Private WithEvents mf_Tasks As frmTasks  '---все задания, связанные с frmTasks
Private WithEvents mf_Task As frmTask    '---отдельное задание, связанное с frmTask
Private mo_DataGrid As DataGrid
Private WithEvents mo_DataProvider As clsDataProvider '---основные данные
Private ml_CurTaskID As Long             '---выбранный идентификатор задания
Private ms_Project As String             '---применяется с frmProject
Private mo_ProgConfig As clsProgConfig
Private ms_TaskFilter As String
```

```
Private mo_Task As clsTask
Private mb_NeedRefresh As Boolean
Private ms_Resource As String
Private ms_Source As String
'---открытые объекты и события
Public Event TaskUpdated()
```

И что мы имеем? Много комментариев, среди которых нет ни одного, который описывал бы назначение центрального объекта. *Плохой код и бездарный кот!* И каким же образом человек со стороны сможет понять, зачем этот объект нужен, если нет никаких описаний?! Для этого придется основательно изучать код. Я-то его знаю вдоль и поперек, а вот свежий взгляд наткнется на труднопреодолимое препятствие.

Будь вы менеджером, вы бы, конечно, настояли на введении заголовка модуля с указанием его автора и даты создания. Кроме того, вы, вероятно, потребовали бы от программиста составить обзор модуля, обозначив в нем имена открытых процедур и механизм «жизнеобеспечения» ими объекта.

Нельзя, однако же, не признать некоторые достоинства вышеприведенного фрагмента кода. Обратите внимание на имена переменных — `m` в них идентифицирует область действия (модуль), а следующий символ обозначает тип переменной (`l` соответствует длинной переменной, `s` — строковой, `b` — логической, и т. д.).

Теперь рассмотрим конкретную процедуру, инициирующую процесс отображения формы задания:

```
Public Sub Show(Optional sResource As String = "")
  If (mf_Tasks Is Nothing) Then
    SetHourglass
    Set mf_Tasks = New frmTasks
    Load mf_Tasks
    Set mo_DataGrid = mf_Tasks.grdTasks
    '---load tasks
    LoadTaskGrid
    '---Load resource combo
    mo_PickList.LoadPickList mf_Tasks.cboResource, PIC_RESOURCE
    '---configure task list
    ms_Resource = sResource
    mf_Tasks.Configure ms_Resource
    If sResource <> "" Then '---установка источника данных
                            для отображения отдельного задания
      ms_TaskFilter = "Assigned =" & Chr$(39) & sResource & Chr$(39)
      mo_DataProvider.Filter ms_TaskFilter
      mo_DataProvider.Sort "Status"
    End If
    SetReady
  End If
  With mf_Tasks
    .WindowState = 0
    .Show
    .ZOrder 0
  End With
End Sub
```

Здесь, несмотря на немногочисленность комментариев по именам вызываемых процедур, можно определить их назначение — таким образом, в некотором отношении этот код можно признать самодокументированным. С другой стороны, комментарии модульного уровня опять же отсутствуют — о том, что это ключевая подпрограмма с открытой областью действия, код ничего не говорит.

Как насчет связности и взаимозависимости?

Что касается связности, то здесь вам придется поверить мне на слово, поскольку, как мы выяснили в предыдущем разделе, комментариев в коде не хватает и разобраться в общем процессе исполнения и структуре кода довольно сложно. Но, вставив комментарий насчет связности и взаимозависимости, я исправился. Вам же полагается знать следующее.

Программа управляется объектом clsApplication с глобальной областью действия. Из него создаются и управляются на высоком уровне все остальные объекты. К примеру, в процедуре с понятным именем clsApplication.StartApplication создана родительская форма MDI и ряд других вспомогательных объектов. С точки зрения процесса исполнения программы это сугубо положительный момент.

Объявления модулей в clsApplication выглядят так:

```
Private WithEvents mf_Parent As mdiManager
Private WithEvents mo_Projects As clsProjects
Private WithEvents mo_Tasks As clsTasks
Private WithEvents mo_Today As clsToday
Private WithEvents mo_Archive As clsArchive
Private mo_Contacts As clsContacts
Private mo_DataService As clsDataService
Private mo_Reports As clsReports
Private mo_ProgConfig As clsProgConfig
Private mo_PickList As clsPickList
Private mc_Tasks As New Collection    '---коллекция объектов clsTasks
Private mc_Projects As New Collection '---коллекция объектов clsProjects
Private mc_Sources As New Collection  '---коллекция объектов clsSource
Private mo_User As clsUser
Private mo_Source As clsSource
Private ms_DSN As String 'применяется при подключении
```

Здесь, если не считать нехватки комментариев уровня модуля, присутствуют все дочерние объекты clsApplication. Из этого фрагмента кода в принципе можно вывести всю объектную иерархию программы.

Реализация в программе многочисленных событий заметно «стройнит» код форм. В большинстве случаев родителями форм выступают модули классов, причем имена форм явственно свидетельствуют об этих отношениях. К примеру, clsProjects запускает frmProjects; впоследствии, если в форме происходит какое-то событие, его аналог сразу запускается в родительском элементе управления. Таким образом, код по большей части локализуется, что, в свою очередь, способствует инкапсуляции.

Все содержащиеся в программе основные объекты (clsTasks, clsProjects и clsToday) обращаются к локальной копии объекта-источника данных под именем clsDataProvider. Это модуль класса VB, в котором поведение источника данных приравнено к значению vbDataSource. Объект clsDataProvider пользуется услугами посредника clsDataService, причем посреднические обязанности осуществляются через функцию под именем GetDataProviderRS, которая одним своим именем демонстрирует факт извлечения набора записей источника данных. Здесь же расположено большинство управляющих программой SQL-операторов. Наконец, приставка «Get» указывает на действие и в некоторой степени объясняет назначение процедуры. По-моему, я уже достаточно пожурил свою живность насчет нехватки комментариев, так что об этом недостатке я больше говорить не буду.

С точки зрения доступа к данным объект clsDataService является открытым интерфейсом логического звена данных. Объект clsDataProvider, описанный в предыдущем абзаце, обеспечивает взаимодействие между данными и уровнем графического пользовательского интерфейса. В составе clsDataService есть дочерний объект под именем clsDataAccess, который фактически осуществляет подключение к базе данных и исполняет передаваемые родительским объектом SQL-операторы. Подобного рода разделение обслуживания сводит взаимозависимость к минимуму и, по моему мнению, существенно облегчает сопровождение и модернизацию электронного администратора.

Другие достоинства и недостатки

По существу, я лишь мельком коснулся некоторых аспектов, которые можно было бы проверить на этом примере. Далее я затрону еще пару вопросов, которые могут вам пригодиться при проверке собственного кода.

Как проводится подключение к базе данных?

Соединение с базой данных устанавливается в начале исполнения, поддерживается в активном состоянии и по мере необходимости передается нуждающимся в данных объектам. Эта схема вполне приемлема для двухзвенной программы с одним пользователем. В то же время, если мы попытаемся масштабировать мою программу для нескольких пользователей, окажется, что обозначенное проектное решение не слишком удачно. Причина, по которой я к нему обратился, заключается в том, что его можно было удобно и быстро реализовать. Оправдывают ли эти преимущества принятые решения? В принципе, мне этого хватает, поскольку расширять свою программу я не собираюсь. С другой стороны, это наглядный пример принятия под давлением временных ограничений простейшего решения, о котором впоследствии можно сильно пожалеть.

Какую роль в моей программе исполняют наборы записей?

Динамические наборы записей открываются в каждом объекте по отдельности и, подобно соединению с базой данных, поддерживаются в активном состоянии. Такое решение повышает реактивность программы, если с ней на одной физической машине работает один пользователь, но сильно ухудшает шансы на масштабирование. Бессвязные наборы записей в данном случае могли бы существенно улучшить возможности масштабирования программы.

Нехватка связей между таблицами — это еще один очевидный недостаток кода, связанный с наборами записей. Иначе говоря, вместо того чтобы задать ссылку внешним ключом на таблицу проектов, я сохраняю имя проекта в таблице заданий. Таким образом, я полностью исключаю возможность переименования проектов. *Плохой код и безмозглый кот!* И, опять же, пожертвовав здесь значительно более оптимальным решением, я предпочел завершить кодирование побыстрее. Впрочем, в защиту выбранного решения выступает и то обстоятельство, что процесс генерации отчетов с помощью программы Crystal Reports при передаче в файл-шаблон ненормализированного набора записей значительно упростился. Кроме того, так мне стало удобнее разрабатывать сам шаблон — ведь, работая в программе Crystal Designer, я просто открывал базу данных и копировал в шаблон ее поля.

Конечно, можно было бы подумать и над другими решениями, но то, на котором я остановился, по крайней мере, позволило оперативно генерировать для подведомственной мне группы еженедельные отчеты.

Есть ли в коде волшебные числа[1]?

Нет. Поскольку мне хотелось сделать программу удобочитаемой, я в основном отдавал предпочтение закрытым и открытым перечислимым типам (перечислимые типы с глобальной областью действия присутствуют в basMain). Есть все же несколько массивов элементов управления, на которых вместо перечислимого типа установлена числовая ссылка. Но в основном, чтобы понять назначение параметра, достаточно взглянуть на его определение.

Заключение

Как вы заметили, большинство недостатков кода я пытаюсь оправдать нехваткой времени. На самом деле это много о чем говорит — думаю, что когда вы будете проводить критические обзоры кода, проблема времени окажется одной из самых существенных. Но ведь это вы руководитель, и поэтому темп разработки, при котором можно достичь приемлемого качества, всецело зависит от вашей воли. Поскольку в данном проекте я выступал сразу в двух ролях — руководителя и кодировщика, — можете считать, что я завалил проект во всех отношениях. Если у вас есть желание озадачить меня какими-то соображениями, связанными с моим кодом, отправляйте вопросы и претензии по адресу herdingcats@mindspring.com.

Я вам советую иногда перечитывать главу 6 — так вы хотя бы не забудете основные принципы технического руководства. Написание этой главы, честно говоря, побудило меня сознаться в собственных неудачах в роли руководителя, и я в очередной раз осознал опасность неумения применять на практике полученные знания.

---

[1] Волшебными я называю числа, которые внезапно начинают играть серьезную роль, — по той лишь причине, что найти определение их значений мне не удается.

Библиография

Ресурсы для специалистов по выпасу котов

В библиографию я поместил все работы, упоминаемые в сносках, и ряд других трудов по теме книги — в основном те, которые недавно опубликованы.

Книги в категории «Разработка программного обеспечения» снабжены краткими аннотациями — надеюсь, так вам будет легче определиться с тем, стоит ли читать каждое конкретное издание. По-моему, если книги читаются, их не может быть слишком много. Естественно, у каждого должны быть свои критерии отбора, и хочется верить, что с моей скромной помощью вы сможете выбрать самые полезные и приятные для чтения источники.

Разработка программного обеспечения

Книги в этом разделе расположены согласно моему субъективному мнению о том, насколько они значимы для специалистов нашей области. Впрочем, они все в равной степени заслуживают вашего внимания.

Классические труды

- **Brooks, Frederick P. The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition. New York: Addison-Wesley, 1995.**

  Основываясь на собственном опыте деятельности в качестве руководителя проекта IBM 360, Брукс излагает принципы разработки, казавшиеся в его время новаторскими и полностью доказавшие со временем свою состоятельность. Его хрестоматийный принцип — «стоит ввести в запаздывающий проект новых людей, как он начнет запаздывать еще больше» — подтвержден многократно. Обязательное чтение для всех руководителей. С точки зрения технологии данные Брукса устарели, но проведенный им анализ задач персонала для групп разработчиков полностью актуален.

- **Weinberg, Gerald M. The Psychology of Computer Programming, Silver Anniversary Edition. New York: Dorset House Publishing, 1998.**

  Еще одна классическая работа в нашей области. Вейнберг, до сих пор продолжающий активные исследования, этой своей книгой 1971 года перевернул

представления о разработке программных средств. Он одним из первых провел углубленный анализ человеческого фактора и руководства в программировании, объединив тем самым технические аспекты процесса разработки с психологией. Многие принципы, изложенные в этом издании, на сегодняшний день не менее актуальны, чем тридцать лет назад.

Выдающиеся работы

- **Beck, Kent. Extreme Programming Explained: Embrace Change. Reading, MA: Addison-Wesley, 2000.**

  Бек часто обращается к школе программной инженерии, но при этом предлагает новые методы и практические рекомендации для тех, кто окончательно погряз в процессе кодирования. С точки зрения руководителя его методология кажется несколько диковатой, однако нельзя не признать, что она способствует интеграции небольших групп разработчиков. Тем кто предпочитает писать код в одиночку, работа не понравится.

- **Cockburn, Alistair. Agile Software Development. New York: Addison-Wesley, 2002.**

  Термин «гибкая разработка» появился не так уж давно, но описанные в этой замечательной работе методы в полной мере адаптированы к реальным повседневным попыткам перенести бизнес-требования на реализацию программных продуктов. Если в ходе разработки область действия разрастается (а разве бывает по-другому?), прочтите эту книгу — она научит адекватно реагировать на такого рода изменения.

- **DeMarco, Tom, and Timothy Lister. Peopleware: Productive Projects and Teams, Second Edition. New York: Dorset House Publishing, 1999.**

  Переиздание любой книги по компьютерной тематике неизменно свидетельствует о ее качестве. Этот труд, написанный двумя профессиональными руководителями программных проектов, — лишнее тому подтверждение. В работе рассматриваются вопросы пространственного планирования, формирования рабочих групп, руководства специалистами и многие другие вопросы, способствующие упорядочиванию деятельности ведущих программистов.

- **Highsmith, James A., III. Adaptive Software Development. New York: Dorset House Publishing, 2000.**

  Будьте уверены — этой книге уготована долгая жизнь. Воспринимать изложенный в ней материал временами трудновато, но лишь потому, что Хайсмит заставляет читателя глубже анализировать характер процессов разработки программных средств. Подзаголовок книги гласит: «Сотрудничество в руководстве сложными системами». Автор логично доказывает сложность повседневной деятельности руководителей проектов и программистов. Из-за одних лишь его пассажей о сотрудничестве издание стоит приобрести. Работа замечательно дополняет труд Кокберна по той же теме.

- **Hunt, Andrew, and David Thomas. The Pragmatic Programmer. New York: Addison-Wesley, 2000.**

  Прекрасная книга! Заставьте прочесть ее всех программистов в своей группе. Справедливость советов, которые дают ее авторы, подтверждается годами опы-

та; не сомневаюсь, что по мере ознакомления с ней вы устанете кивать в знак согласия. Предложенная авторами практическая методика управления основывается на предотвращении хаотичности в разработке программных средств.

- **McBreen, Pete. Software Craftsmanship. New York: Addison-Wesley, 2002.**

  Смелыми и аргументированными рассуждениями Макбрин обеспечил своей книге устойчивые позиции среди литературы по программной инженерии. Несмотря на подзаголовок «Новые веяния», автор утверждает, что мастерство кодирования знаменует собой возврат к истокам разработки программных средств. Если вы согласны с тем, что кодирование есть не что иное, как искусство, эта книга для вас. Если вы не согласны, все равно прочтите эту книгу — возможно, она вас переубедит.

- **McCarthy, Jim. Dynamics of Software Development. Redmond, WA: Microsoft Press, 1995.**

  Написанная хорошим языком и далекая от поверхностного взгляда книга, в которой рассказывается о том, на какие ухищрения идут группы разработчиков, лишь бы поставить программный продукт вовремя. Здесь отражены многие проблемы, которые компании Microsoft в разное время пришлось разрешать самостоятельно. Автор пишет с увлечением, глубоким пониманием проблемы и прицелом на практическую деятельность. Кроме того, Маккарти излагает материал в категориях гибкости, и здесь нужно иметь в виду, что когда эта книга появилась, термин «гибкость» еще не был так распространен, как сегодня. Можете рассказать об этой книге на следующем производственном совещании.

Примечательные работы

- **Brown, William H., et al. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. New York: John Wiley & Sons, 1998.**

  Эта и следующая книги доходчиво доказывают педагогические преимущества ложных приемов. В книге анализируются проблемы, с которыми вы наверняка уже сталкивались. Кроме того, в ней изложены многочисленные полезные методики разрешения трудностей в кодировании.

- **Brown, William H., et al. AntiPatterns in Project Management. New York: John Wiley & Sons, 2000.**

  Эта книга, ориентированная на анализ методик управления с позиций негативных эталонов, органично дополняет мои соображения относительно «обратной стороны» деятельности руководителей (см. главу 7). Серьезно помогает в оценке собственного стиля руководства.

- **Constantine, Larry L. Beyond Chaos: The Expert Edge in Managing Software Development. New York: Addison-Wesley, 2001.**

  Сборник статей, написанных разными авторами, но, тем не менее, вышедших под именем Ларри Константайна. Тематический диапазон — от вопросов, связанных с персоналом, до процессов и соблюдения сроков. Менеджменту и коллективной работе посвящен целый раздел.

- **Constantine, Larry L. The Peopleware Papers. Upper Saddle River, NJ:Yourdon Press, 2001.**

  Концептуально Константайн относится к тому же лагерю, что и Демарко с Листером (см. раздел «Выдающиеся работы»). Работа представляет собой сборники статей, опубликованных за несколько лет в ведущих журналах, посвященных разработке программных продуктов. Книгу можно начинать читать с любой страницы. Особо занимательны взгляды автора на руководство специалистами.

- **Glass, Robert L. ComputingFailure.com. Upper Saddle River, NJ: Prentice Hall, 2001.**

  Подборка рассказов Гласса о несостоятельных проектах. Акцент сделан на банкротствах в период бума интернет-компаний. Прекрасное изложение материала и дельные прогнозы по поводу связи незавершенки с объемом инвестиций.

- **Glass, Robert L. Software Runaways. Upper Saddle River, NJ: Prentice Hall, 1998.**

  Уже много лет Гласс информирует общественность о несостоятельных программных проектах и весьма в этом преуспел. Анализ он проводит с точки зрения программной инженерии, а о последних «достижениях» пишет увлекательно и осведомленно.

- **Maguire, Steve. Writing Solid Code. Redmond, WA: Microsoft Press, 1993.**

  Акцент в этой книге делается на достижение безошибочности кода C, однако изложенные в ней принципы применимы к любому языку. Работа сопоставима с упомянутым в этом разделе изданием Макконнелла. Они обе написаны примерно в одно время и прекрасно дополняют друг друга.

- **Malveau, Raphael, and Thomas J. Mowbray. Software Architect Bootcamp. Upper Saddle River, NJ: Prentice Hall, 2001.**

  Книга эта входит в серию Всемирного института программных архитекторов (Worldwide Institute of Software Architects, WWISA). Прекрасно разъясняет существо программной архитектуры для тех, кто не отличает ее от проектного решения.

- **McConnell, Steve. Code Complete. Redmond,WA: Microsoft Press, 1993.**

  Хрестоматийный труд о программировании на C. Практические рекомендации сохраняют справедливость и сейчас, в контексте современных проблем кодирования — взять хотя бы раздел о соглашениях по именованию. Все, что сообщает Макконнелл, помогает писать самодокументированный код.

- **Olson, Don S., and Carol L. Stimmel. The Manager Pool: Patterns for Radical Leadership. New York: Addison-Wesley, 2002.**

  Если вам интересно сравнить собственные навыки руководства программистами с опытом других специалистов, эта книга — для вас. Различные модели руководства разбираются с психологической точки зрения. Особой похвалы авторы заслуживают за классификацию универсальных поведенческих моделей, которых придерживается большинство руководителей программных продуктов.

- **Sewell, Marc T., and Laura M. Sewell. The Software Architect's Profession: An Introduction. Upper Saddle River, NJ: Prentice Hall, 2002.**

  Еще одно наименование из серии WWISA. Авторами выступили президент этого института вместе со своей женой — специалистом в области клинической

психологии. Весьма занимательный труд, читается на одном дыхании. Роль программного архитектора сравнивается с профессией «настоящего» архитектора.

- **Yourdon, Edward. Death March. Upper Saddle River, NJ: Yourdon Press, 1999.**

  Еще один классический труд. Всем нам приходилось участвовать в «умерших» проектах, и всякий раз мы не понимали, почему они такими становятся. Йордон объясняет не только причины этого явления, но и способы реанимации проекта (правда, дело это довольно сложное). Подзаголовок «Универсальное руководство разработчика программного обеспечения по реанимации «умерших» проектов» говорит сам за себя.

- **Yourdon, Edward. Decline & Fall of the American Programmer. Englewood Cliffs, NJ: Yourdon Press, 1993.**

  Первое издание этой книги, написанной одним из ведущих специалистов в нашей области, оказало эффект разорвавшейся бомбы. С тех пор многие рассматриваемые в этом труде Йордона темы получили продолжение в работах других авторов. Заголовок немного обескураживает, но содержание актуально по сей день. Во многих университетах эта книга даже введена в учебный план.

- **Yourdon, Edward. Rise & Resurrection of the American Programmer. Upper Saddle River, NJ: Yourdon Press, 1996.**

  Продолжение предыдущей работы Йордона, причем не менее значимое, чем первая часть. Обширный опыт Йордона позволяет ему с легкостью анализировать различные тенденции. Прекрасно дополняет более пессимистичный труд, изданный несколькими годами ранее.

Полезные работы

- **Bullock, James, Gerald M. Weinberg, and Marie Benesh, eds. Roundtable on Project Management. New York: Dorset House Publishing, 2001.**

  Сборник дискуссий о руководстве проектами, проходивших на разных веб-сайтах. Весьма занимательные свидетельства о том, как выявить трудности в выполнении проекта с точки зрения персонала и процесса. Помогает избежать подобных ситуаций в собственных проектах.

- **DeMarco, Tom. The Deadline. New York: Dorset House Publishing, 1997.**

  Димарко — один из лучших специалистов, исследующих проблемы кадрового обеспечения программных проектов. Конкретно эта книга написана в жанре романа. Об управлении проектами говорится таким языком, что оторваться трудно.

- **Freedman, Daniel P., and Gerald M. Weinberg. Handbook of Walkthroughs, Inspections, and Technical Reviews. New York: Dorset House Publishing, 1990.**

  Книга в значительной степени устарела, но, с другой стороны, проблема, которой она посвящена, очень мало исследована. В частности, здесь изложен подробный план проведения технических обзоров.

- **Humphrey, Watts S. Managing the Software Process. New York: Addison-Wesley, 1989.**

  Классический труд. Прочитать его стоит всем — даже тем, чьи взгляды не совпадают с позицией Хэмфри, который исходит из принципов школы программ-

ной инженерии. Несмотря на то что с момента выхода книги прошло 15 лет, многие изложенные в ней универсальные принципы до сих пор актуальны; кроме того, любопытно сравнить предлагаемые в ней принципы с принципами других методологий разработки.

- **Jones, Capers. Applied Software Measurement. New York: McGraw-Hill, 1991.**

  Книга из серии Института программной инженерии (Software Engineering Institute, SEI). (Кстати, в той же серии вышла предыдущая книга Уотса Хэмфри.) Ее предмет — контроль качества программных средств — не так очевиден, как кажется на самом деле. Вместо того чтобы мыслить по стереотипу «если оно работает, значит, можно выпускать», советую ознакомиться с изложенными в этой книге принципами и впредь перед компиляцией уделять больше внимания вопросам тестирования.

- **Kerth, Norman L. Project Retrospectives. New York: Dorset House Publishing, 2001.**

  Если вам нужен современный справочник по критическому обзору проектов, считайте, что вы его уже нашли. Автор — ведущий специалист в рассматриваемой области, хотя публикуется довольно редко. На мой взгляд, в книге много дельных и обдуманных предложений.

- **Whitehead, Richard. Leading a Software Development Team: A Developer's Guide to Successfully Leading People & Projects. New York: Addison-Wesley, 2001.**

  Уайтхеду удалось написать доступную по стилю изложения книгу, ориентированную на начинающих руководителей программных проектов. Составленная по модели «вопрос-ответ», она очень удачно раскрывает многие из тех проблем, с которыми все мы сталкиваемся в деле выпаса котов.

Общие работы по менеджменту

- Carlson, Richard. Don't Sweat the Small Stuff at Work. New York: Hyperion, 1998.
- Champy, James, and Nitin Nohria. The Arc of Ambition. New York: Perseus Books, 2000.
- Covey, Stephen R. Principle-Centered Leadership. New York: Simon & Schuster, 1992.
- Covey, Stephen R. The 7 Habits of Highly Effective People. New York: Simon & Schuster, 1989.
- Grove, Andrew S. Only the Paranoid Survive. New York: Random House, 1996.
- Katzenbach, Jon R., and Douglas K. Smith. The Wisdom of Teams. New York: HarperCollins, 1999.
- Pasternack, Bruce A., and Albert J. Viscio. The Centerless Corporation. New York: Simon & Shuster, 1998.
- Welch, Jack. Straight from the Gut. New York: Warner Business Books, 2001.

Работы по языкам программирования

- Appleman, Dan. Moving to VB .NET: Strategies, Concepts, and Code. Berkeley, CA: Apress, 2001.
- Foxall, James D. Practical Standards for Microsoft Visual Basic. Redmond, WA: Microsoft Press, 2000.
- Hollis, Billy S. Visual Basic 6 Design, Specification, and Objects. Upper Saddle River, NJ: Prentice Hall, 1999.

Разные работы

- Blake, William. The Complete Poetry and Prose of William Blake. Edited by David V. Erdman. Berkeley, CA: University of California Press, 1982.
- Elliot, T. S. Collected Poems 1909-1962. New York: Harcourt Brace Jovanovich, 1971.
- Kauffman, Stuart. At Home in the Universe. New York: Oxford University Press, 1995.
- Kranz, Gene. Failure Is Not an Option. New York: Simon & Schuster, 2000.
- Levinson, Daniel J. The Seasons of a Man's Life. New York: Ballantine Books, 1978.
- Merton, Thomas. No Man Is an Island. New York: Harcourt Brace Jovanovich, 1955.
- Raymond, Eric S., ed. The New Hacker's Dictionary, Third Edition. Cambridge, MA: The MIT Press, 1998.
- Rich, Adrienne. The Dream of a Common Language. New York: W. W. Norton & Company, 1978.
- Shenk, David. The End of Patience: Cautionary Notes on the Information Revolution. Bloomington, IN: Indiana University Press, 1999.
- Strathern, Paul. Turing and the Computer. New York: Doubleday, 1997.
- Tzu, Sun. The Art of War. New York: Dell Publishing, 1983.
- Wallace, James, and Jim Erickson. Hard Drive: Bill Gates and the Making of the Microsoft Empire. New York: John Wiley & Sons, 1992.
- Ullman, Ellen. Close to the Machine. San Francisco: City Lights Books, 1997.
- Yeats, William Butler. Selected Poems and Three Plays of William Butler Yeats. Edited by M. L. Rosenthal. New York: Collier Books, 1986.

Алфавитный указатель

Б

В

Г

Д

Е

Ж

З

И

Й

К

Л

М

Н

О

П

P

C

Т

У

Ф

Х

Ц

Ч

Ш

Э

Дж. Ханк Рейнвотер

Как пасти котов. Наставление для программистов, руководящих другими программистами

Перевел с английского Ю. Гороховский

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*В. Шрага*
Литературный редактор	*А. Жданов*
Художник	*Л. Адуевская*
Иллюстрации	*Л. Родионова*
Корректоры	*А. Моносов, В. Листова*
Верстка	*Л. Родионова*

ООО «Питер Пресс», 192102, Санкт-Петербург, ул. Андреевская (д. Волкова), д. 3, литер А, пом. 7Н.
Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12.000 — Книги печатные профессиональные, технические и научные.
Подписано в печать 21.07.15. Формат 70×100/16. Усл. п. л. 20,640. Тираж 300. Заказ 0000
Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87