# JOEL ON SOFTWARE

And on Diverse and
Occasionally Related Matters
That Will Prove of Interest
to Software Developers,
Designers, and Managers,
and to Those Who,
Whether by Good Fortune
or Ill Luck, Work with Them
in Some Capacity

*Joel Spolsky*

Apress®

ПРОФЕССИОНАЛЬНО

# Джоэл о программировании

и разнообразных и иногда родственных вопросах, которые должны быть интересны разработчикам программного обеспечения, проектировщикам и менеджерам, а также тем, кому посчастливилось или не повезло в каком-то качестве работать с ними

*Джоэл Спольски*

СИМВОЛ®

*Санкт-Петербург — Москва*
*2008*

Серия «Профессионально»

Джоэл Спольски

# Джоэл о программировании

**и разнообразных и иногда родственных вопросах, которые должны быть интересны разработчикам программного обеспечения, проектировщикам и менеджерам, а также тем, кому посчастливилось или не повезло в каком-то качестве работать с ними**

Перевод С. Маккавеева


| | |
|---|---|
| Главный редактор | *А. Галунов* |
| Зав. редакцией | *Н. Макарова* |
| Научный редактор | *О. Цилюрик* |
| Редактор | *В. Овчинников* |
| Художник | *В. Гренда* |
| Верстка | *О. Макарова* |
| Корректор | *И. Губченко* |

*Спольски Дж.*

Джоэл о программировании и разнообразных и иногда родственных вопросах, которые должны быть интересны разработчикам программного обеспечения, проектировщикам и менеджерам, а также тем, кому посчастливилось или не повезло в каком-то качестве работать с ними. – Пер. с англ. – СПб.: Символ-Плюс, 2008. – 352 с., ил.

ISBN-10: 5-93286-063-4

ISBN-13: 978-5-93286-063-2


Книга представляет собой подборку эссе, опубликованных автором на его сайте *http://www.joelonsoftware.com*. Талант и глубокое проникновение в суть предмета сделали Джоэла мастером своего дела, а остроумие и едкий юмор принесли сайту скандальную известность среди программистов. Затронуты практически все вообразимые аспекты создания ПО от лучших способов устройства рабочего места программиста до лучших способов написания программного кода. Издание адресовано широкому кругу читателей – и тем, кто собирается руководить программистами, и самим программистам – как приверженцам Microsoft, так и сторонникам открытого кода.


**ISBN-10: 5-93286-063-4**

**ISBN-13: 978-5-93286-063-2**

**ISBN 1-59059-389-8 (англ)**





Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, edit@symbol.ru. Лицензия ЛП N 000054 от 25.12.98.
Налоговая льгота – общероссийский классификатор продукции
ОК 005-93, том 2; 953000 – книги и брошюры.
Подписано в печать 30.10.2007. Формат 70x90$^{1}/_{16}$ . Печать офсетная.
Объем 22 печ. л. Доп. тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

*Моим родителям, воспитавшим меня в убеждении,*
*что все взрослые пишут книги.*

# Оглавление

# Об авторе

**Джоэл Спольски**, ветеран программной индустрии, ведет веблог «Joel on Software» (Джоэл о программировании) (*www.joelonsoftware.com*) – один из самых независимых и популярных среди программистов сайтов. Сайт Джоэла назвали «манифестом анти-Дильберта». Спольски спроектировал и написал программы, используемые миллионами людей, работал над рядом продуктов – от Microsoft Excel до пользовательского интерфейса Juno. Он основал компанию Fog Creek Software, расположенную в Нью-Йорке.

# Благодарности

Я хотел бы поблагодарить своего издателя, редактора и друга Гэри Корнелла, благодаря которому стала возможной эта книга. Сотрудники Apress всегда были любезны и добры. Apress – это редкое компьютерное издательство, в котором стремятся прежде всего заботиться об авторе, и я в восторге от совместной работы с ними.

Но еще до книги был сайт, обязанный своему существованию как бесплатному хостингу и рекламе, обеспеченным Дэйвом Винером (Dave Winer) из UserLand Software, так и его энтузиазму. Я признателен Филипу Гринспену (Philip Greenspun), который убедил меня, что знаниями надо делиться и публиковать их в Интернете, чтобы и другие могли это узнать. И Ною Тратту (Noah Tratt), вселившему в меня вдохновение, сказав однажды, что некоторые свои проповеди я должен предать бумаге.

Хочу поблагодарить своих коллег из бригады Microsoft Excel, которые на старте моей карьеры научили меня тому, как разрабатывать коммерческие приложения.

За истекшие годы мой сайт посетили миллионы людей, и примерно 10 000 из них нашли время, чтобы написать мне чудесное письмо. Эти ободряющие послания – единственная причина, по которой я продолжал писать, и хотя мне не хватит места, чтобы перечислить всех поименно, я очень ценю эти письма.

# Введение

Ты никогда не *стремился* стать менеджером. Как и большинство разработчиков программ, с которыми я знаком, ты был бы гораздо счастливее, если бы тебе позволили спокойно сидеть и писать код. Но ты лучший разработчик, и когда с Найджелом, прежним руководителем группы, произошел этот *несчастный* случай на банджи и с лэптопом, всем показалось естественным, что на его место надо выдвинуть тебя, звезду команды.

И вот теперь у тебя собственный кабинет (вместо одной клетки на двоих с вечным летним стажером), и ты должен заполнять все эти оценки эффективности труда, составляемые каждые полгода (вместо того чтобы с удовольствием портить себе зрение, глядя целый день в экран), если, конечно, ты не тратишь попусту время, разбирая причудливые требования примадонн программирования, фамильярно хлопающих по спине ребят из отдела продаж, творчески настроенных «конструкторов UI» (приглашавшихся, вообще-то, в качестве графических дизайнеров), которым нужны сверкающие кнопки OK/Cancel, способные *отражать* (простите, какое значение RGB для цвета «отражающий»?). И искать ответы на глупые вопросы старшего вице-президента, который почерпнул все свои знания о программном обеспечении из статьи в журнале, издаваемом Delta Airlines для своих пассажиров. «Почему мы используем Oracle, а не Java? Я слышал, что он более унифицирован».

Добро пожаловать в администрирование! Знаете, что я вам скажу? Управление программными проектами не имеет *никакого* отношения к программированию. Если вы до сих пор не занимались ничем, кроме написания кода, то вам предстоит открыть, что человеческие существа несколько менее предсказуемы, чем обычный процессор Intel.

Во всяком случае прежний руководитель группы, Найджел, никогда не преуспевал в этом. «Я не собираюсь стать одним из тех руководителей, которые проводят все свое время в бессмысленных совещаниях, – любил он говорить, и в этом была не только бравада. – Я думаю, что все-таки смогу 85% времени заниматься кодированием и только немножко – *администрированием*».

На самом деле Найджел *хотел* сказать другое: «У меня вообще нет *никакого* понятия о том, как руководить этим проектом, и я надеюсь, что если я просто буду продолжать кодировать так, как занимался этим до того, как меня назначили ответственным, то все как-нибудь само устроится». Оно, конечно, не устроилось, что в значительной мере и объясняет, почему Найджел прыгал на банджи вместе с IBM ThinkPad в тот злополучный день.

И Найджелу еще здорово повезло, что он выздоровел, с учетом всех обстоятельств, и сейчас он работает техническим директором небольшой компании WhatTimeIsIt.com, которую он создал вместе со своими друзьями по банджи, и у него есть всего шесть месяцев, чтобы сдать совершенно новую систему, созданную с чистого листа, и больше ему не удастся прикинуться больным.

~

Управление программными проектами не слишком хорошо изучено. Не существует степеней в управлении программными проектами, и не так много книг написано на эту тему. Кто-то из тех, кто работал над действительно удачными программными проектами, разбогател и ушел на покой разводить форель на фермах, не воспользовавшись возможностью передать накопленный ими опыт следующему поколению, а многие другие прогорели и нашли себе менее напряженную работу типа преподавания корректива английского языка хулиганам из городского гетто.

В результате многие программные проекты проваливаются, явно или скрыто, потому что ни у кого в команде нет представления о том, как должен управляться успешный программный проект. Поэтому очень многие команды так никогда и не выпускают свой продукт, или делают его слишком долго, или выпускают продукт, который никому не нужен. Но хуже всего то, что эти люди несчастны и ненавидят каждую потраченную на него минуту. Жизнь слишком коротка, чтобы ненавидеть свою работу.

Пару лет назад я опубликовал на своем сайте «тест Джоэла» – список из двенадцати характерных признаков успешно управляемых команд раз-

работчиков. В их числе такие вещи, как ведение базы данных по ошибкам, предложение поступающим на работу написать код во время собеседования и т. д. (не волнуйтесь, я подробно расскажу об этом). Поразительно, но множество людей написало мне, что их команда заработала всего лишь два или три очка из двенадцати.

Два или три!

Это почти сюрреализм. Представьте себе бригаду столяров, пытающихся делать мебель и не имеющих никакого представления о шурупах. Они употребляют исключительно гвозди и загоняют их в дерево с помощью туфель для чечетки, потому что никто не рассказывал им о существовании молотков.

Управление программными проектами требует совершенно иных навыков и приемов, нежели написание кода; это два совершенно различных и не связанных между собой поля. Написание кода так же отличается от управления проектом, как нейрохирургия от выпекания кренделей. Нет никаких оснований полагать, что у блестящего нейрохирурга, каким-то образом попавшего на производство кренделей в результате некоего разрыва в пространственно-временном континууме, окажется хоть малейшее представление о том, как делать эти самые крендели, даже если он *окончил* Гарвардский медицинский колледж. Но люди, однако, продолжают думать, что можно взять ведущего программиста и без какой-либо переподготовки перевести его в администраторы.

Так же как и вышеупомянутый нейрохирург, ты и Найджел были переведены на новую работу, в администраторы, где требуется контактировать – о, Боже! – с *людьми*, а не с компиляторами. И если тебе казалось, что современные компиляторы Java полны ошибок и непредсказуемы, то ты должен просто подождать, пока возникнет первая проблема с программистом-примадонной. Управление командами, состоящими из людей, заставит тебя смотреть на шаблоны C++ как на явно *элементарную* вещь.

Тем не менее *есть* приемы, позволяющие руководить успешными программными проектами. Это искусство шагнуло дальше гвоздей и туфель для чечетки. У нас в распоряжении молотки, отвертки и торцовочные пилы, умеющие снимать двойную фаску. В этой книге я поставил перед собой задачу познакомить читателя со всеми приемами, которые смог вспомнить, на всех уровнях – от составления графика работы руководителем команды до выработки конкурентоспособной стратегии исполнительным директором. Вы узнаете:

- Как набирать и мотивировать к работе самых лучших работников. (Это самый важный фактор успешного программного проекта.)
- Как делать реальные оценки и графики, и зачем они нужны.
- Как проектировать функции ПО и писать спецификации, которые действительно полезны для работы, а не для подшивок «один раз написал и можно не читать», из которых надстраивают перегородки в офисе.
- Как избежать распространенных ловушек в разработке ПО, и почему программисты всегда неправы, настаивая на том, что надо «все выбросить и начать сначала».
- Как организовывать команды и стимулировать их работу, и почему программистам нужны офисы с закрывающимися дверями.
- Когда следует писать собственный код с чистого листа, даже если есть почти пригодная версия, которую можно загрузить из Интернета.
- Почему всегда кажется, что программные проекты застопориваются через первую пару месяцев работы.
- Что означает стратегия ПО, и почему BeOS была обречена с первого же дня.
- А также многое другое.

Книга весьма субъективна. Ради краткости я был вынужден опустить слова типа «по моему мнению» в начале каждого предложения, потому что, как оказалось, каждое предложение в этой книге представляет мое личное мнение. И она не всеобъемлюща, но, пожалуй, сможет послужить хорошей отправной точкой.

## *А, так вы были на моем веб-сайте...*

Значительная часть содержимого этой книги впервые увидела свет в виде статей на моем веб-сайте, «Джоэл о программировании» (*www.joelonsoftware.com*), где я записываю свои мысли на протяжении последних лет. Книга, которую вы держите в руках, значительно более *связна*, как я надеюсь, чем сайт (под *связностью* я подразумеваю возможность читать, лежа в ванной, не подвергаясь риску поражения электрическим током).

Для удобства чтения мы разделили книгу на три основные секции. Первая секция посвящена разработке программного обеспечения в небольших масштабах: что надо делать, чтобы выпускать ПО, безопасное для человека. Во второй части собран ряд статей об управлении программиста-

ми и командами программистов. Третья часть составлена несколько произвольно, но в основном посвящена созданию устойчивого программного бизнеса. Вы узнаете, почему bloatware всегда побеждает, чем Ben & Jerry's отличается от Amazon, и я попытаюсь доказать, что методологии разработки программного обеспечения обычно служат признаком низкой квалификации работников.

В книге есть материал, посвященный и другим темам, но вы можете сами открыть ее и начать читать.

ГЛАВА ОДИННАДЦАТАЯ

# Тотальное уничтожение ошибок

31 ИЮЛЯ 2001 ГОДА, ВТОРНИК

Качество программного обеспечения, а точнее отсутствие оного вызывает жалобы у всех. Теперь у меня своя собственная компания, и я вынужден как-то решать эту проблему. В последние две недели мы в Fog Creek занимались только подготовкой к выпуску новой инкрементной версии FogBUGZ с намерением устранить все известные ошибки (числом около 30).

Для разработчика программного обеспечения исправление ошибок – это дело правое. Так? Ведь это всегда правое дело?

Нет!

Исправлять ошибки необходимо только тогда, когда сохранение ошибки обходится дороже, чем ее исправление.

Измерить стоимость того и другого непросто, но можно. Приведу пример. Допустим, что вы управляете фабрикой по производству сэндвичей с арахисовым маслом и желе. Ваша фабрика выпускает 100 000 сэндвичей в сутки. Недавно благодаря появлению некоторых новых сортов (чесночно-арахисовое масло с острым соусом хабанеро) спрос на вашу продукцию поднялся выше крыши. Выпуская 100 000 сэндвичей, фабрика работает на полную мощность, но спрос, вероятно, ближе к 200 000. Вы просто не в состоянии произвести больше. Каждый сэндвич дает вам прибыль в 15 центов. Стало быть, вы ежедневно теряете 15 000 долларов потенциальной прибыли из-за отсутствия достаточных мощностей.

Строить еще одну фабрику – слишком дорого. У вас нет такого капитала, и вы опасаетесь, что мода на пряные/чесночные сэндвичи может закончиться. Но вы продолжаете терять по 15 000 в день.

К счастью, вы взяли на работу Джейсона. Джейсон – программист 14-ти лет от роду, который поковырялся в компьютерах, управляющих производством, и теперь ему кажется, что он знает, как увеличить скорость конвейера вдвое. Начитался на Slashdot про разгон чипов. Проведенное испытание показало, что вроде бы это возможно.

Есть только одно обстоятельство, удерживающее вас от того, чтобы дать делу полный ход. Из-за какой-то *крохотной,* ничтожной, жалкой ошибочки примерно раз в час производственная линия заминает один сэндвич. Джейсон хочет исправить эту мелкую ошибку. Он считает, что ему будет достаточно трех дней. Что вы сделаете – дадите ему возможность исправить ошибку или запустите программу в теперешнем состоянии, с известной ошибкой?

Откладывание запуска программы на три дня обойдется в 45 000 долларов упущенной прибыли. А ваша экономия составит стоимость сырья для 72 сэндвичей. (Джейсон в любом случае исправит ошибку через 3 дня.) Ну, не знаю, сколько стоят сэндвичи *на вашей* планете, но здесь, на Земле, они намного дешевле 625 долларов.

Так о чем это я говорил? Ах, да. Иногда *исправлять ошибку невыгодно.* Вот другой пример: в вашей программе есть ошибка, приводящая к ее аварийному завершению при открытии очень больших файлов, но она может возникнуть лишь у единственного пользователя, который работает в OS/2 и у которого, как вам известно, больших файлов не бывает. Ну и нечего ее исправлять. Бывают вещи и похуже. Еще я перестал думать о тех, у кого мониторы с 16 цветами или кто когда-то поставил Windows 95 и не обновлял ее 7 лет. Можете мне поверить, эти люди не станут тратить деньги на коробочные программные продукты.

Но обычно ошибки исправлять надо. Даже «безвредные» ошибки портят репутацию компании и продукта, которая в конечном итоге сильно повлияет на ваши доходы. Избавиться от репутации компании, выпускающей продукт с ошибками, трудно. Если вам все же надо выпустить продукт версии .01, то вот некоторые соображения о том, как искать и исправлять «*правильные*» ошибки, т.е. такие, исправление которых экономически оправданно.

## *1. Обеспечьте получение информации об ошибках*

В случае FogBUGZ мы прибегли для этого к двум способам. Во-первых, мы перехватываем все ошибки на нашем бесплатном демосервере, со-

бираем как можно больше информации о них и отправляем все это электронной почтой бригаде разработчиков. В результате мы обнаружили массу ошибок, что было очень хорошо. Например, мы выяснили, что ряд людей не вводит дату в нужном месте экрана «Fix For». Тут даже не было сообщения об ошибке, просто программа аварийно завершалась (что для веб-приложения означает получение гадкого сообщения IIS вместо ожидаемой страницы).

Когда я работал с Juno, у нас была еще более замечательная система автоматического получения данных ошибок «прямо с поля боя». С помощью TOOLHELP.DLL мы установили обработчик, который при каждом аварийном завершении Juno не умирал, пока стек сбрасывался в log-файл, и только потом сходил в могилу. Когда программа в очередной раз соединялась с Интернетом, чтобы отправить почту, она загружала log-файл. На стадии бета-тестирования мы собирали эти файлы, сравнивали все аварийные завершения и вводили их в базу данных ошибок. В результате обнаружились буквально сотни ошибок, вызывающих аварийное завершение. Когда у вас миллион пользователей, можно придти в *полное изумление* от того, какой именно код может приводить к аварии, часто из-за очень малого объема доступной памяти или очень плохих компьютеров (вы напишете Packard Bell без ошибок?). Например, в коде

```
int foo( object& r )
{
   r.Blah();
   return 1;
}
```

возникал фатальный сбой из-за того, что ссылка r оказывалась равной NULL, хотя это совершенно невозможно – *не бывает* нулевых ссылок, C++ это гарантирует. Можете мне не верить, но если, имея миллионы пользователей, подождать некоторое время и скрупулезно собирать дампы их стеков, то обнаруживаются фатальные сбои в таком вот коде, вы их видите, но не верите своим глазам. (Их нельзя исправить, потому что в них виноваты космические лучи. Купите новый компьютер и больше *не* устанавливайте в панель задач никаких красивых шароварных штуковин, которые вам попадутся.)

Кроме того, абсолютно все обращения в службу технической поддержки проверяются на предмет, не связаны ли они с наличием ошибки. Приняв вызов, мы пытаемся определить, что можно сделать для его аннулирования. Например, в прежней Setup для FogBUGZ предполагалось, что Fog-

BUGZ будет выполняться под учетной записью анонимного пользователя Интернета. В 95 процентах случаев это было оправданно, а в 5 процентах предположение было неверным, но каждый из этих 5 процентов случаев заканчивался звонком в нашу службу поддержки. Поэтому мы модифицировали Setup, чтобы он запрашивал имя учетной записи.

## *2. Обеспечьте себе экономичную обратную связь*

Не всегда можно *точно* определить, во что обойдется исправление каждой ошибки, но кое-что сделать *можно*: предъявить счет на «стоимость» технической поддержки производственному подразделению. В первой половине 1990-х в Microsoft началась финансовая реорганизация, в результате которой каждое производственное подразделение должно было возмещать полную стоимость всех обращений за технической поддержкой. В результате производственные подразделения стали настаивать, чтобы PSS (служба технической поддержки Microsoft) регулярно предоставляла им «горячую десятку» ошибок. Когда разработчики сосредоточились именно на этих ошибках, стоимость поддержки продуктов резко упала.

Это несколько противоречит современной тенденции, когда отдел технической поддержки оплачивает собственное функционирование, как это происходит в большинстве крупных компаний. В Juno предполагалось, что техническая поддержка будет содержать себя, взимая плату с клиентов. Перемещая экономическую ответственность за ошибки на самих пользователей, вы лишаетесь даже ограниченных возможностей определить наносимый им ущерб. (Взамен вы получаете разгневанных пользователей, возмущенных тем, что они должны платить за *ваши* ошибки, и рассказывающих об этом своим друзьям, так что трудно сказать, во сколько это обойдется вам в итоге. Справедливости ради надо сказать, что собственно Juno был бесплатным продуктом, поэтому перестаньте ругаться.)

Способ примирить одно с другим состоит в том, чтобы *не* взимать с пользователя денег, если обращение в службу поддержки было вызвано ошибкой в вашем продукте. Microsoft так и делает, и это правильно, и я ни разу не платил за звонок в Microsoft! Зато счет на 245 долларов, или сколько там стоит в наше время один случай, предъявляется производственному отделу. В результате их прибыль от проданного вам продукта улетучивается, что создает совершенно правильные экономические стимулы. Это напоминает мне, почему игры для DOS были *жутким* бизнесом... Чтобы они прилично выглядели и быстро работали, обычно требовались необычные

видеодрайверы, и одно-единственное обращение в службу поддержки по поводу видеодрайверов могло съесть прибыль от *20* экземпляров вашего продукта, если Egghead, Ingram и то объявление на MTV еще не поглотили *всех* ваших накоплений.

## *3. Посчитайте, во что вам обойдется исправить их все*

Наша Fog Creek Software – крохотная компания (конечно, мысленно мы представляем себе ее иначе), и команда разработчиков сама же принимает все обращения за технической поддержкой. Это обходилось нам примерно в час времени ежедневно и в соответствии с нашими тарифами на консультирование стоило примерно 75 000 долларов в год. Нам казалось, что мы сможем уменьшить это время до 15 минут в день, если исправим все известные ошибки.

По очень грубым прикидкам это означает, что чистая приведенная экономия составит примерно 150 000 долларов. Это оправдывает 62 дня работы; если вы сможете сделать это быстрее, чем за 62 человеко-дня, стоит браться за работу.

С помощью удобной функции оценки, имеющейся в FogBUGZ, мы подсчитали, что исправить все ошибки можно за 20 человеко-дней (двое сотрудников будут работать две недели), т. е. «потратив» 48 000 и получив взамен 150 000, и это будет очень неплохая инвестиционная прибыль *благодаря экономии на одной только технической поддержке*. (Заметьте, что вместо тарифа на консультационные услуги можно было взять величину зарплат программистов и накладных расходов и получить тот же результат 3:1, поэтому они равны.)

Я даже не стал считать прибыль от получения более совершенного продукта, но можно сделать и это. Старый код в течение одного месяца июля дал нам 55 аварий на демо-сервере, у 17 разных пользователей. Можно представить себе, что хотя бы *один* из них решил не покупать FogBUGZ из-за ошибок при работе демо-версии (точной статистики у меня нет). Во всяком случае, объем потерянных продаж мог составить от 7000 до 100 000 долларов в приведенном значении. (Если заняться этим серьезно, то можно получить и реальные цифры.)

Еще один вопрос. Можно ли назначать более высокую цену за продукт на основании того, что в нем меньше ошибок? Это еще больше увеличило бы ценность устранения ошибок. Мне кажется, что в исключительных слу-

чаях количество ошибок влияет на цену, но мне затруднительно привести в подтверждение пример из области коробочных продуктов.

## *Только не бейте!*

Кто-то, прочитав эссе вроде этого, несомненно придет к нелепым выводам типа «Джоэл не советует исправлять ошибки». На самом деле я считаю, что в большинстве случаев исправление ошибок экономически оправданно. Но иногда можно получить более высокую прибыль *иными* способами, не занимаясь тотальным истреблением ошибок. Если приходится выбирать между исправлением ошибки для того малого с OS/2 и добавлением новой функции, благодаря которой можно продать 20 000 экземпляров программы в General Electric, то тогда, друг с OS/2, извини. Если же упрямство *все-таки* заставляет тебя считать, что исправить ошибку с OS/2 важнее, чем добавить функцию для GE, то не исключено, что конкуренты, которые думают иначе, вытеснят тебя с рынка.

Тем не менее в душе я остаюсь оптимистом и верю, что выпуск продукции очень высокого качества таит в себе получение высокой прибыли, которую не так просто оценить. Ваши сотрудники будут больше гордиться своим трудом. Меньше ваших клиентов вернет вам обратно по почте CD с вашим продуктом, предварительно изжарив его в микроволновке и порубив топором. Поэтому я склоняюсь к качеству (мы действительно исправили в FogBUGZ *все известные ошибки*, а не только самые заметные) и уверен, что благодаря полному устранению ошибок с демосервера мы создали чрезвычайно надежный продукт.

ГЛАВА ДВЕНАДЦАТАЯ

# Пять миров

6 мая 2002 года, понедельник

Первым местом моей постоянной работы был хлебозавод. В одном огромном помещении пекли хлеб, а в другом упаковывали его в коробки. В первом целый день все мучились с тестом. Оно застревало в механизмах, прилипало к рукам, волосам и обуви, и у каждого был маленький скребок для краски, которым это тесто соскабливали. Во втором помещении все целый день мучились с крошками: они застревали в механизмах и в волосах, и у каждого была маленькая кисточка. Мне подумалось, что в каждой работе есть свое зло – собственный источник вечных неприятностей, и я порадовался, что не работал на фабрике бритвенных лезвий.

В программировании есть свои источники зла, и в одном помещении они не такие, как в другом, но почему-то какую бы книгу о разработке программного обеспечения вы ни взяли, вряд ли найдете в ней упоминания о разных помещениях, не говоря уже о существующих разновидностях зла.

Большую часть своей послехлебозаводской профессиональной деятельности я занимался созданием программного обеспечения, предназначенного для использования миллионами людей. Когда я впервые прочел, что в «экстремальном программировании» считается обязательным иметь в составе бригады «клиента», я пришел в недоумение. При работе над почтовым клиентом для Juno, крупного общенационального интернет-провайдера, у нас были *миллионы* клиентов, включая великое множество милых старушек, о которых ничего не говорится в литературе, посвященной формированию команд программистов.

Когда мне доводилось выступать с лекциями перед корпоративными программистами и я начинал распространяться перед ними о том, что у нас в Microsoft было так-то и так-то, они смотрели на меня, как на инопланетянина: «Джоэл, здесь не Microsoft, у нас *нет* миллионов клиентов, у нас только Джим во вспомогательном офисе.»

В конце концов я разобрался. Существуют разные сферы разработки программ, и в каждой из них действуют свои законы. А этот кажущийся очевидным факт долго игнорировался. Да, все мы разработчики программного обеспечения, но когда мы не обращаем внимания на различия между нашими сферами, начинаются неуместные войны .NET против Java или методологий ускоренной разработки против программных технологий. Вы читаете статью о моделировании UML, но там нигде не сказано, что оно бессмысленно при программировании драйверов устройств. Или вы читаете в телеконференции, что 20-мегабайтный исполняемый модуль .NET не представляет никаких проблем, но там не сказано, что если надо написать код для пейджера с 32 килобайтами EPROM, то это уже проблема, да еще какая.

Я считаю, что можно выделить пять сфер:

- Коробочные продукты
- Внутрифирменное ПО
- Встроенное ПО
- Игры
- Одноразовые программы

## *1. Коробочные продукты*

Я называю так (shrinkwrap) все продукты, с которыми будут «самостоятельно» работать многие люди. Эти продукты могут быть упакованы в целлофан и продаваться в CompUSA либо загружаться через Интернет. Эти программы могут быть коммерческими, условно-бесплатными (shareware), с открытым кодом, GNU или с какими-либо другими условиями распространения; главное в том, что их установят и будут использовать тысячи и миллионы людей. Возможно даже, что это веб-сайт, посещаемый тысячами или миллионами людей.

Жизнь разработчиков коробочных продуктов определяется двумя вечными неприятными факторами:

- У их программ неисчислимое количество пользователей, часто имеющих альтернативы. Поэтому для успеха программы интерфейс пользователя должен быть проще, чем обычно. Половину времени они тратят на то, чтобы *еще больше упростить* жизнь своей деревенской тетушке.
- Их программы выполняются на многих компьютерах, поэтому код должен быть предельно устойчивым к различиям между машинами. Обеспечив это требование, они тратят *оставшееся* время на то, чтобы разглядеть метки версий DLL.

На прошлой неделе я получил письмо об ошибке в продукте CityDesk, выпускаемом моей компанией. Эта ошибка проявляется только с польскими клавиатурами, потому что на них для ввода специальных символов применяется так называемая клавиша AltGr (никогда об этом не слышал). Мы тестировали Windows 95, 95OSR2, 98, 98SE, Me, NT 4.0, Windows 2000, и Windows XP Home и Pro. Мы тестировали с установленными IE 5.01, 5.5 или 6.0. Мы тестировали версии Windows, локализованные для английского, испанского, французского, иврита и традиционного китайского языков. Но мы еще *не совсем* добрались до польского.

Есть три главные разновидности коробочного продукта: с открытым исходным кодом (open source), веб-приложения и консалтинговое ПО.

### *Приложения с открытым исходным кодом*

Эти программы часто разрабатываются бесплатно, что сильно влияет на динамику. Например, в команде, состоящей исключительно из добровольцев, часто не делают то, что считается «неинтересным», что, как красноречиво объяснил Мэттью Томас (Matthew Thomas), наносит явный ущерб юзабилити.[1] Разработчики очень часто географически удалены друг от друга, что совершенно меняет характер обмена информацией между ними. В мире open source очень редко происходит личное общение, когда рисуют прямоугольники и стрелки на доске, поэтому проектные решения, для которых полезно вычерчивать квадратики и стрелки, в таких проектах удаются обычно плохо. В итоге географически распределенные команды гораздо больше преуспевают в клонировании уже готового программного обеспечения, когда проектирование если и требуется, то в незначитель-

[1] Matthew, Thomas (Мэттью, Томас) «Why Free Software Usability Tends to Suck» (Почему тошнит от юзабилити бесплатного ПО), 2002, *http://mpt. phrasewise.com/2002/04/13.*

ном объеме. Но по большей части программное обеспечение все же рассчитано на самостоятельное выполнение всеми желающими, поэтому я причисляю его к коробочному.

### *Веб-приложения*

Такое программное обеспечение, как Hotmail, eBay или даже контентный сайт, тоже должно быть простым в использовании и работать во многих броузерах. У большинства пользователей установлен «броузер-монополист», но больше всего шума от пользователей, у которых по неизвестным причинам стоит какой-нибудь доисторический броузер, не способный правильно показывать соответствующий стандартам HTML, даже если в приложение *встроить* самого Джеффри Зельдмана. Поэтому в итоге масса времени тратится на поиск в Google таких вещей, как «ошибка шрифта CSS в Netscape 4.72». Хотя в данном случае у разработчиков есть возможность хотя бы частично контролировать среду «развертывания» – компьютеры в центре обработки данных, но им приходится иметь дело с большим разнообразием броузеров и большим количеством пользователей, поэтому я считаю, что по существу это разновидность коробочного продукта.

### *Консалтинговое ПО*

Эта разновидность коробочного продукта требует такого объема работы при настройке и установке, что его запуск должна обеспечивать целая армия консультантов, стоящая огромных денег. К этой категории часто относятся системы CRM и CMS. Возникает подозрение, что само по себе это программное обеспечение не может *делать* ничего, оно лишь служит предлогом привести к вам эту армию консультантов, выставляющих счета в размере 300 долларов за час работы. Консалтинговое ПО маскируется под коробочное, но высокая цена его реализации делает его более близким к *заказному* ПО.

## *2. Заказное программное обеспечение*

Внутрифирменное ПО предназначено для работы только в конкретных условиях одной компании. Это значительно облегчает его разработку. Можно довольно точно оценить среду, в которой оно будет выполняться. Можно потребовать установить конкретную версию Internet Explorer, или Microsoft Office, или Windows NT 4.0 с *шестым* сервис-паком. Если вам нужна диаграмма, пусть ее строит Excel: у каждого в отделе установлен Ex-

cel. (Но попробуйте так поступить в коробочном продукте, и вы лишитесь половины потенциальных клиентов.)

Корпоративные разработчики не любят признаваться, что в программном обеспечении, разрабатываемом собственными силами, юзабилити уделяется небольшое внимание, поскольку работать с этим ПО будут люди, у которых нет другого выбора, им приходится работать с тем, что дадут, да и количество их невелико. И ошибки в таком ПО обычно встречаются чаще, чем в коробочном. Здесь важнее скорость разработки. Поскольку стоимость разработки ложится на бюджет лишь одной компании, объем ресурсов, которые можно привлечь для разработки, оказывается значительно меньше. Это Microsoft может позволить себе потратить $200 000 000 на разработку операционной системы, которая обычному человеку обойдется всего в $98. Но когда Чаттануга Рэйлроуд разрабатывает внутрифирменную трейдинговую платформу чучу, надо проверить, оправданны ли такие расходы для одной компании. Чтобы получить приемлемую прибыль от инвестированного капитала, разработчики внутрифирменного ПО не могут позволить себе тратить столько, сколько можно было бы потратить на разработку коробочного продукта. Фактически одно из главных отличий внутрифирменного ПО от коробочного состоит в том, что начиная с некоторого момента дополнительные усилия, направленные на повышение его надежности или простоты эксплуатации, становятся резко убыточными; для коробочного же продукта последний 1 процент, потраченный на повышение стабильности и простоты использования, может обеспечить решающее преимущество в конкуренции. Поэтому, как ни печально, большинство внутрифирменного ПО весьма плохо делает даже то, что должно было бы делать идеально. Это может угнетающе воздействовать на молодых и полных энтузиазма разработчиков, которых заставляют прекратить работу над ПО, поскольку оно уже «достаточно хорошее».

## *3. Встроенное ПО*

Уникальность этого ПО в том, что оно является частью какой-либо аппаратуры и, как правило, не обновляется. (Даже если это *технически* возможно. Уверяю вас, что никто не загружает обновление флэш-памяти своей микроволновой печи). Это совершенно особый мир. Требования к качеству очень высоки, потому что исправить ошибку уже нельзя. Программы выполняются процессором, который обычно гораздо слабее, чем в обычном настольном компьютере, поэтому масса времени тратится на

оптимизацию и ручную настройку. Скорость выполнения кода существеннее, чем его элегантность. Доступность устройств ввода-вывода ограничена. В системе GPS машины, которую я арендовал на прошлой неделе, были такие жалкие устройства ввода-вывода, что юзабилити ее производила тяжелое впечатление. Никогда не пробовали ввести адрес в устройство без клавиатуры? Или искать дорогу по карте размером с часы Casio? Но я отклонился от темы.

## *4. Игры*

Игры уникальны в двух отношениях. Во-первых, экономика разработки игр ориентирована на создание хитов. Некоторые игры становятся хитами, но гораздо большее их количество проваливается, и если вы хотите зарабатывать деньги на производстве игр, то должны это учитывать и создать портфель игр в расчете, что один блокбастер компенсирует убытки всех остальных. Это больше похоже на создание кинокартин, чем программ.

Более серьезная проблема разработки игр в том, что у них только одна версия. Сыграв до конца в *Duke Nukem 3D* и убив большого босса, никто не собирается делать обновление до *Duke Nukem 3.1D*, в котором будут исправлены некоторые ошибки и добавлено новое оружие. Это скучно. Поэтому в играх существуют такие же требования к качеству, как во встроенном ПО, и крайняя финансовая необходимость добиться успеха с первого раза, тогда как у разработчиков коробочного ПО, возящихся со своими DLL, есть хотя бы надежда, что если версия 1.0 дрянь и ее никто не берет, то 2.0 будет лучше, и ее станут покупать.

## *5. Разовые программы*

Пятая область, упоминаемая для полноты, это одноразовый код. Это временный код, который пишется, чтобы получить что-то еще, и становится ненужным, как только оно получено. Например, вам может понадобиться небольшой сценарий оболочки, чтобы перевести полученный файл входных данных в формат, который необходим для каких-то других целей, и такая операция оказывается разовой. В одноразовом коде не таится никаких опасностей, хотя существует закон, согласно которому разовый код может внезапно стать внутрифирменным и какой-нибудь обнаруживший его бизнес-администратор решит, что на этом можно развернуть

целый бизнес, – и *hate*! Появляется еще одна консалтинговая компания, предлагающая непрочные «решения масштаба предприятия».

## *Свой мир надо знать*

Мы живем в разных мирах. Ну и что? Какое это может иметь значение? А вот какое. Когда вы читаете некую книгу о программных методологиях, написанную профессиональным гуру/консультантом в области разработки программного обеспечения, то скорее всего речь в ней идет о внутрифирменной разработке корпоративного программного обеспечения. Не коробочных продуктов, не встроенного ПО и уж явно не игр. Почему? Потому что нанимают этих гуру корпорации. Они платят деньги. (Поверьте мне, id Software не станет приглашать Эда Йордона, чтобы он рассуждал о структурном анализе.)

В последнее время я внимательно присматривался к экстремальному программированию (XP). Для многих типов проектов XP и другие ускоренные методы оказываются глотком свежего воздуха по сравнению с неестественными, скучными и порочными «процессами», применяемыми не в одной из софтверных компаний, и не вызывает сомнения, что такие технологии, как разработка на основе тестирования, просто потрясают, если вы сумеете их применить. Но в этом и загвоздка: *если вы сумеете их применить*. В XP есть целый ряд подводных камней, которые могут стать опасными в конкретной ситуации. Например, никто не добился успеха в применении разработки на основе тестирования для создания GUI. Я построил целый сервер веб-приложения с помощью разработки на основе тестирования, и он работал потрясающе, потому что сервер приложения занят в основном обработкой строк. Но я не смог применить *ни один вид* автоматизированного тестирования для создания моих GUI. В лучшем случае можно воспользоваться автоматизированным тестированием, которое не поможет, если надо реализовать буксировку методом drag and drop.

XP также предполагает простоту выполнения рефакторинга, особенно при наличии тестов, подтверждающих его корректность. В монолитных проектах, особенно разрабатываемых внутри фирм и независимо от внешних факторов, так оно и есть. Но я *не могу* произвольно изменить схему продукта FogBUGZ, производимого моей компанией, потому что многие наши клиенты написали для него свой код, а если я сделаю его неработоспособным, они не станут покупать следующую версию, а тогда я не смогу заплатить программистам, и весь этот карточный домик рухнет. Тщатель-

ным анализом и проектированием нельзя пренебрегать, если вы собираетесь корректно взаимодействовать с другими. Вот я и пытаюсь выяснить, как применять лучшие из ускоренных методологий в разработке коробочного ПО и определять, когда это невозможно.

Как правило, законы разработки ПО одинаковы для любых типов проектов, но не всегда. Когда кто-нибудь рассказывает о методологии, надо подумать о том, применима ли она в *вашей конкретной* работе. Учитывайте то, в какой области работает этот человек. Все мы можем чему-то поучиться друг у друга. В *каждой сфере есть свои источники трудностей*. Если вы корпоративный разработчик приложений для сервера Java, то не стоит вводить людей в заблуждение и объяснять причину отсутствия у вас «кошмара DLL» тем, что Java сам по себе более совершенный язык программирования. Вы живете без кошмара DLL, *потому что ваша программа работает только на одной машине*. Если вы разрабатываете игры, перестаньте жаловаться на то, как медленно работает интерпретируемый байт-код. *Он не для вас*. Он для внутрифирменных разработчиков, программу которых пытаются «завалить» четыре оператора, ведущих бухгалтерию. Уважайте и понимайте проблемы теста/хлебных крошек, которые есть у каждого, и давайте сделаем Usenet более цивилизованной средой обитания!

ГЛАВА ДВАДЦАТЬ ДЕВЯТАЯ

# Рик Чэпмен в поисках глупости[1]

1 августа 2003 года, пятница

Во всех известных мне хайтек-компаниях идет война между гиками и пиджаками.

Прежде чем вы станете читать новую пропагандистскую книгу волшебника маркетинга программного обеспечения и сверхпиджака Рика Чэпмена, послушайте меня минутку и узнайте, что думают гики.

Минутка-то у вас найдется?

Представьте себе самого типичного бледного, пьющего джолт-колу, питающегося китайской кухней, играющего в видеоигры, читающего Slashdot, не вылезающего из командной строки Linux чудака. Поскольку это всего лишь стереотип, можете представлять себе его как угодно, коротышкой или пухленьким мальчиком, но в любом случае он *не из тех*, кто играет в футбол с ребятами из своей школы, когда приезжает к маме на День Благодарения. Кроме того, поскольку это стереотип, мне не надо долго извиняться за то, что я сделал его *таким*.

Вот как рассуждает наш стереотипный программер: «Microsoft делает более слабые продукты, но проводит более сильный маркетинг, поэтому все покупают у них».

Спросите его, как он относится к маркетинговым специалистам в его собственной компании. «Они просто тупы. Вчера в комнате отдыха я круп-

---

[1] Эта глава первоначально вышла в виде предисловия к книге Меррила Рика Чэпмена (Merrill Rick Chapman) «In Search of Stupidity: Over 20 Years of High-Tech Marketing Disasters» (В поисках глупости: 20 с лишним лет маркетинговых провалов высоких технологий), Apress, 2003.

но поспорил с этой глупой девицей из отдела продаж, и через десять минут выяснилось, что она *понятия не имеет*, какая разница между 802.11*a* и 802.11*b*. Фе!»

А скажи мне, милый друг, чем занимаются люди из маркетинга? «Без понятия. Наверное, играют с клиентами в гольф, когда не заставляют меня исправлять их идиотские таблицы спецификаций. Если бы это зависело от меня, всех бы уволил».

Славный малый по имени Джеффри Тартер ежегодно публиковал список 100 крупнейших производителей программного обеспечения для персональных компьютеров, который он называл «Soft-letter 100». Вот как выглядел этот список в 1984 году:[1]

| Место | Компания | Годовой доход |
|---|---|---|
| #1 | Micropro International | $60 000 000 |
| #2 | Microsoft Corp. | $55 000 000 |
| #3 | Lotus | $53 000 000 |
| #4 | Digital Research | $45 000 000 |
| #5 | VisiCorp | $43 000 000 |
| #6 | Ashton-Tate | $35 000 000 |
| #7 | Peachtree | $21 700 000 |
| #8 | MicroFocus | $15 000 000 |
| #9 | Software Publishing | $14 000 000 |
| #10 | Broderbund | $13 000 000 |

Отлично, Microsoft на втором месте, но она входит в группу компаний с примерно одинаковыми годовыми доходами.

А вот тот же список для 2001 года:

| Место | Компания | Годовой доход |
|---|---|---|
| #1 | Microsoft Corp. | $23 845 000 000 |
| #2 | Adobe | $1 266 378 000 |
| #3 | Novell | $1 103 592 000 |
| #4 | Intuit | $1 076 000 000 |
| #5 | Autodesk | $926 324 000 |

[1] Jeffrey Tarter, *Soft*letter*, April 30, 2001, 17:11.

| Место | Компания | Годовой доход |
|---|---|---|
| #6 | Symantec | $790 153 000 |
| #7 | Network Associates | $745 692 000 |
| #8 | Citrix | $479 446 000 |
| #9 | Macromedia | $295 997 000 |
| #10 | Great Plains | $250 231 000 |

О-о! Соблаговолите отметить: *все до единой компании*, исключая Microsoft, исчезли из первой десятки. Кроме того, обратите, пожалуйста, внимание, что Microsoft *настолько опережает* следующего по результатам игрока, что это даже не смешно. Adobe удвоила бы свою прибыль, если бы прибавила к ней то, что в Microsoft тратят на газированную воду.

Рынок программного обеспечения для персональных компьютеров – это и есть Microsoft. Доходы Microsoft, как можно посчитать, составляют 69% *всех 100 ведущих компаний, вместе взятых*.

Вот об этом мы здесь и говорим.

Это всего лишь более сильный маркетинг, как заявляет наш воображаемый гик? Или это результат противозаконной монополии? (Что вызывает вопрос: как Microsoft *удалось получить* такую монополию? Что-нибудь одно из двух.)

Рик Чэпмен утверждает, что ответ проще: Microsoft была единственной компанией в списке, ни разу не сделавшей глупой фатальной ошибки. Обязана ли она своим успехом интеллектуальному превосходству или лишь слепой удаче, но самой крупной ошибкой Microsoft была танцующая скрепка. И так ли уж плоха она была *на самом деле*? Мы смеялись над ней, отключали ее и все равно пользовались Word, Excel, Outlook и Internet Explorer ежеминутно и ежедневно. Но для всех других компаний, когда-то имевших лидерство на рынке и безвозвратно утративших его, можно указать одну или две крупных ошибки, обернувшихся потерей правильного курса и столкновением с айсбергом. Micropro начала возиться с переделкой архитектуры принтера, вместо того чтобы совершенствовать свой главный продукт WordStar. Lotus потратила полтора года, чтобы втиснуть 123 в машины с 640 килобайтами памяти; когда они справились с этим, появилась Excel, а от машин с памятью 640 Кбайт остались слабые воспоминания. Digital Research невероятно завысила цену на CP/M-86 и утратила

шанс стать стандартом де факто операционных систем для PC.[1] VisiCorp досудилась до собственного исчезновения. Ashton-Tate не упустила возможности прогнать разработчиков dBase, отравив непрочную экологию, столь необходимую для успеха поставщика платформы.

Конечно, я программист и склонен винить за эти глупые ошибки тех, кто занимается маркетингом. Почти все эти ошибки связаны с неспособностью людей бизнеса, не сведущих в технических вопросах, понять элементарные факты технологии. Когда преуспевший с Pepsi Джон Скалли стал продвигать Apple Newton, он не знал того, что известно каждому, кто получил образование в информатике: распознавание рукописного текста *невозможно*. Это происходило в то самое время, когда Билл Гейтс затаскивал программистов на совещания и упрашивал создать единый элемент управления rich text edit («расширенного редактирования текста»), который можно было бы повторно использовать во всех их продуктах. Посади Джима Манци (пиджака, который позволил бизнес-администраторам захватить Lotus) на такое совещание, он стал бы хлопать там глазами. «Что еще за элемент rich text edit?» Ему никогда не пришло бы в голову осуществлять техническое руководство, потому что он не «грокал» технологии; на самом деле само слово *grok* в этом предложении, вероятно, ошеломило бы его.

Если хотите знать *мое* мнение, а оно предвзятое, то *никакая софтверная компания не может добиться успеха*, если у руля в ней не стоит программист. Все живые до сих пор свидетельства подтверждают мой вывод. Но и сами программисты наделали массу тупых ошибок. Поразительное решение Netscape переписать заново свой броузер вместо того, чтобы совершенствовать старый код, обошлось им потерей нескольких лет в Интернете, в течение которых принадлежавшая им доля рынка сократилась с 90% до 4, и это была *идея программиста*. Конечно, ничего не понимавшее в технике неопытное руководство этой компании не понимало, *почему* это плохое решение. Тем не менее есть масса программистов, защищающих решение Netscape переписать все с самого начала. «Джоэл, старый код был действительно ужасен!» Да-да, ну-ну. Такими программистами нужно восхищаться за их любовь к хорошо сделанному коду, но их на 100 метров

[1] Как мы все знаем, причиной того, что «стандартом де факто операционных систем для PC стала MS-DOS», а не гораздо более совершенная CP/M-86, было не только (и не столько) то, что Digital Research невероятно завысила цену, но и коррумпированная и «личностная» возня внутри IBM периода принятия этого решения. Можно предполагать, что мы еще при своей жизни узнаем и более глубинные «геополитические» причины такого развития событий... – *Примеч. науч. ред.*

нельзя подпускать к принятию каких-либо деловых решений, потому что очевидно, что ясность кода для них важнее, чем, простите, выпуск продукта на рынок.

Поэтому я немного уступлю Рику и скажу, что если вы хотите добиться успеха в софтверном бизнесе, то ваши руководители должны хорошо знать и любить программирование, но они должны знать и любить бизнес тоже. Найти лидера с большими способностями в обоих этих направлениях трудно, но это единственный способ избежать одной из тех фатальных ошибок, которые Рик так любовно собрал в своей книге. Так что прочтите ее, посмейтесь, и, если вашей компанией руководит какая-нибудь тупая голова, приведите в порядок свое резюме и начните подыскивать жилище в Редмонде.