ПРОФЕССИОНАЛЬНО

Дэвид Платт

СОФТ ОТСТОЙ!

И ЧТО С ЭТИМ ДЕЛАТЬ?

СИМВОЛ®

Why Software Sucks...

and what you can do about it

David S. Platt

Addison-Wesley

ПРОФЕССИОНАЛЬНО

Софт – отстой!

И что с этим делать?

Дэвид С. Платт

СИМВОЛ®

Санкт-Петербург — Москва
2008

Серия «Профессионально»

Дэвид С. Платт

Софт – отстой! И что с этим делать?

Перевод С. Маккавеева

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Редактор	*В. Овчинников*
Художник	*В. Гренда*
Корректор	*О. Макарова*
Верстка	*О. Макарова*

Платт Д.

Софт – отстой! И что с этим делать? – Пер. с англ. – СПб: Символ-Плюс, 2007. – 248 с., ил.

ISBN-10: 5-93286-097-9
ISBN-13: 978-5-93286-097-7

Дэвид Платт, за плечами которого 20-летний опыт программирования и преподавания, утверждает, что современное ПО – отстой. Оно не защищено и позволяет программам злоумышленников проникать из Интернета в наши компьютеры. Оно ненадежно и ломается в самый ответственный момент, уничтожая плоды наших долгих трудов и не давая средств к их спасению. Им трудно пользоваться, потому что приходится ломать голову над тем, как выполнить простейшие операции. Дэвид Платт объясняет, почему программы могут так разочаровывать и даже оказываться опасными, а также предлагает несколько способов борьбы с этим. Изложение содержит много примеров и сдобрено юмором. Не многие компьютерные книжки способны заставить громко смеяться. Но Дэйв не только смешит, он делится очень интересными наблюдениями и взглядами, излагая их в ясном и занимательном стиле. Он призывает пользователей сообща бороться с производителями плохого ПО и приглашает единомышленников на свой сайт *www.suckbusters.com*.

ISBN-10: 5-93286-097-9
ISBN-13: 978-5-93286-097-7
ISBN 0-321-46675-6 (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 22.11.2007. Формат 70x90$^{1}/_{16}$. Печать офсетная.
Объем 15,5 печ. л. Тираж 2000 экз. Заказ N
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Моим дочерям Люси Катрине Платт
и Аннабел Роуз Платт

Оглавление

Похвальные отзывы

«Я только что закончил читать лучшую компьютерную книжку [«Why Software Sucks»] и хочу сказать о ней доброе слово. Включите ее в свой список обязательного чтения, если у вас есть программы, если вы любите программы, не любите программистов или даже сами программист, потому что мистер Платт (который преподает программирование) вознамерился сбить спесь с тех, кто считает, что если он может написать программу, то может сделать ее простой в обращении… Книга смешная, но она также служит важным предупреждением тем софтверным компаниям, которые намереваются сократить свои расходы на поддержку клиентов. Если вас мучает вопрос «Почему хорошие программисты пишут такие ужасные программы?», то ответ на него вы найдете в этой книге.»

— Джон Мак-Кормик (Gohn McCormick),
ведущий колонки Locksmith на TechRepublic.com

«Скажу сразу, не многие компьютерные книжки способны заставить меня громко смеяться. Но Дэйв Платт не только смешит, он также делится некоторыми очень интересными наблюдениями и взглядами, излагая их в ясном и занимательном стиле. Это тоже встречается не часто!»

— Генри Лейтнер (Henry Leitner), зам. декана
по информационной технологии и старший преподаватель
вычислительных наук, Гарвардский университет

«Книжка для всех угнетенных пользователей компьютеров, написанная понятным языком.»

— *Стейси Барателли, парикмахер автора*

«Уникальный подход Дэвида к трепещущим проблемам разработки ПО заставил меня по-новому взглянуть на этот процесс. Если вас заботит качество программ, которыми вы пользуетесь или которые разрабатываете, прочтите эту книгу.»

— *Дэйв Чэппелл (Dave Chappell), глава Chappell & Associates*

«Я начал читать эту книгу на работе, но прекратил, не дочитав до конца первой страницы. Я не в силах был спрятать ухмылку на лице! Наслажусь ей, когда вернусь домой и найду укромное местечко для чтения.»

— *Цукаса Макино (Tsukasa Makino), ИТ-менеджер*

«Дэвид объясняет (причем так, что его понимает даже моя теща), почему программы, которыми мы сегодня пользуемся, могут так разочаровывать и даже иногда оказываться опасными, а также предлагает несколько способов борьбы с этим.»

— *Джим Броссо (Jim Brosseau), Clarrus Consulting Group, Inc.*

Об авторе

Дэвид С. Платт руководит Rolling Thunder Computing (*www.rollthunder.com*), учебно-консультационной фирмой. Имея более чем двадцатилетний опыт работы программистом, он преподает разработку программ в Колледже непрерывного обучения Гарвардского Университета и в различных компаниях по всему миру, часто выступает на конференциях. Он автор девяти книг, в том числе «Introducing Microsoft .NET, Third Edition»[1], «The Microsoft Platform Ahead»[2] и «Understanding COM+» (все вышли в Microsoft Press), а также многочисленных журнальных статей и бюллетеней. В 2002 году Microsoft присудила ему звание «Легенды программирования» (Software Legend). Живет в Ипсуиче, шт. Массачусетс.

---

[1] Дэвид С. Платт «Знакомство с Microsoft .NET». — Пер. с англ. — Русская Редакция, 2001.

[2] Дэвид С. Платт «Платформа Microsoft 2005». — Пер. с англ. — Русская Редакция, 2004.

Благодарности

Всякая книга — результат коллективного труда. При работе над этой у меня была превосходная группа поддержки. В Addison-Wesley все поняли и поддержали идею книги, написанной для пользователей компьютеров, а не для программистов, как обычно. Вместо «это не наш профиль» прозвучало «здорово, это для нас отличная возможность».

В первую очередь благодарю моего редактора Питера Гордона (Peter Gordon), который после первого прочтения заметил, что ему не часто попадаются рукописи по компьютерам, которые заставляли бы громко смеяться. Спасибо и другим сотрудникам Addison-Wesley, в число которых входят Курт Джонсон (Curt Johnson), Ким Бедигхаймер (Kim Boedigheimer), Джули Нагил (Julie Nahil), Одри Дойл (Audrey Doyle), Анна Попик (Anna Popick) и Эрик Гэрулей (Eric Garulay). Я благодарю своего друга и коллегу-автора Дэвида Чэппелла (David Chappell) в первую очередь за то, что он направил меня в Addison-Wesley. Хочу также поблагодарить своего агента Алекса Гласса (Alex Glass) из Trident Media Group и автора Кэтрин Коултер (Catherine Coulter), направившую меня к нему.

Написанное здесь было вдохновлено чтением книги «Complications: a Surgeon's Notes on an Imperfect Science» (Осложнения: заметки хирурга о несовершенстве науки) Атула Гаванди (Atul Gawande), представляющей собой взгляд хирурга на его собственную профессию. Я подумал: «А почему бы мне не написать подобную книгу о моей профессии, разработке программного обеспечения? У меня получится веселее». В его книге

есть глава «9000 хирургов», в которой рассказывается о поведении хирургов на их ежегодной конференции и которая послужила непосредственным источником моей главы «Десять тысяч гиков, помешанных на джолт-коле».

Я должен поблагодарить всех своих друзей, коллег, клиентов и студентов — тех, кто прочел первые наброски и высказал ценное для меня мнение. Я многому учусь, работая с вами. И всех разработчиков, которые пишут хорошие программы, благодаря кому я могу показать что-то со словами: «Вот правильная вещь, вот что мы можем на самом деле». И всем разработчикам отстойного ПО тоже спасибо, потому что без них мне не над чем было бы смеяться, и книга получилась бы пресной.

Наконец, хочу поблагодарить свою семью: жену Линду и дочерей Аннабель Роуз и Люси Катрину.

Введение

Современное программное обеспечение — отстой. Приличных слов, чтобы выразить этот факт, нет. Оно не защищено и позволяет программам-преступникам проникать из Интернета в наши жилища. Оно ненадежно и ломается в самый ответственный момент, уничтожая плоды наших многочасовых трудов и не давая средств к их спасению. Работать с ним трудно, потому что приходится ломать голову над тем, как выполнить простейшие операции.

Наверно, вы не впервые это слышите. Вы же не идиот, хотя небезызвестная серия книг с желто-черной обложкой ставит перед собой цель убедить вас в обратном. Нынешнее программное обеспечение отвратительно, и вы всегда это знали. Поэтому вы и рассмеялись, увидев название книги. И ведь этот смех не был произвольным, правда? Вы же не сказали себе, прочтя название: «Ха, забавно, надо бы посмеяться»? Нет. Заглавие ударило по нервам, как запах, посылающий сигнал через кору к рептильному мозгу, и вы разразились непроизвольным безудержным смехом. Когда к вам возвратилось сознание, вы, возможно, подумали что-то вроде «Здорово! Этот малый (умеющий хорошо выражать свои мысли, с рекомендациями, красивый и, судя по всему, к тому же скромный) говорит дело, выражая на бумаге то, что я всегда думал. Притом вовремя». Так оно и есть. В этой книге посредством нетехнического, свободного от жаргона языка рассказывается, почему возникла такая ситуация и можно ли ее как-то изменить. «Черт возьми, надо купить!

И несколько экземпляров для друзей». Отличная мысль. Разве я не сказал, что вы не идиот?

Пятнадцать, даже десять лет назад люди не пользовались программами в повседневной жизни. Мои родители не посылали электронные поздравительные открытки к дням рождения своих внучек и не рассчитывали получить от меня таким же способом цифровые фотографии. Они подводили сальдо своих чековых счетов в бумажной тетради и вели расписание своих дел чернилами в бумажных календарях с милыми картинками. Те немногие, кто регулярно работал с программами, обычно вынуждены были это делать по роду своей работы, например туристические агенты пользовались системами бронирования авиабилетов. У них были маленькие специализированные приложения, для которых требовались дорогое фирменное оборудование, продолжительное обучение и текущая поддержка, а какое-либо применение их для других задач было невозможно. Всемирная паутина World Wide Web была технически сложным изобретением кабинетных ученых. Большинству даже не было известно о ее существовании, не говоря уже о возможности красть с ее помощью музыку или загружать непристойные картинки.

Все совершенно изменилось — по меркам общественного развития практически за ночь, и довольно незаметным для нас образом. Мои родители теперь пользуются для управления чековыми счетами финансовым пакетом, который оплачивает счета электронным способом (поэтому у них никогда не кончаются почтовые марки) и автоматически показывает электронные вклады и оплаченные чеки, когда они поступают в банк. Они теперь хотят не только получать по электронной почте фотографии внучек, но и смотреть живое потоковое видео. Предоставив домашним пользователям неограниченный высокоскоростной доступ примерно за $40 в месяц, Web сделала то, что обещали, но не осуществили поборники атомной энергии — сделать ее слишком дешевой, чтобы стоило мерить количество. Интернет проник всюду настолько глубоко, что штат Пенсильвания убрал с автомобильных номеров свое прозвище «Штат — Краеугольный Камень» (The Keystone State) и заменил его веб-адресом штата *www.state.pa.us* (рис. 1). Флорида пошла дальше Пенсильвании, сохранив прозвище и заменив название штата веб-адресом *MyFlorida.com* (рис. 2). А потребность в туристических агентах сильно сократилась, поскольку у обычного человека есть дешевый и простой доступ к информации, который раньше требовал дорогого и сложного оборудования.

Рис. 1. *Автомобильный номер штата Пенсильвания с веб-адресом вместо прозвища штата*

Рис. 2. *Автомобильный номер штата Флорида с веб-адресом вместо названия штата*

Сегодня мы окружены морями программного обеспечения, но у большинства пользователей нет никакого представления о том, как оно создается и почему оно работает так, а не иначе. Известно только, что оно не очень нам нравится. Каждый может рассказать какую-нибудь жуткую историю, и не одну, связанную с программами, вроде всех этих авиационных кошмаров с многочасовыми ожиданиями в самолете на взлетной полосе или ошибочной отправкой багажа в Тимбукту. Вы весь день работали с некой программой, не зная, что нужно явно сохранять рабочий документ, а потом эта чертова программа грохнулась, и вы все потеряли. А случайная последовательность битов на экране в тот момент была очень похожа на средний палец. С момента выбора в 1982 году журналом «TIME» в качестве «человека года» персонального компьютера репутация последнего опустилась. Карикатура «Деволюция» хорошо отражает чувства многих пользователей (рис. 3).

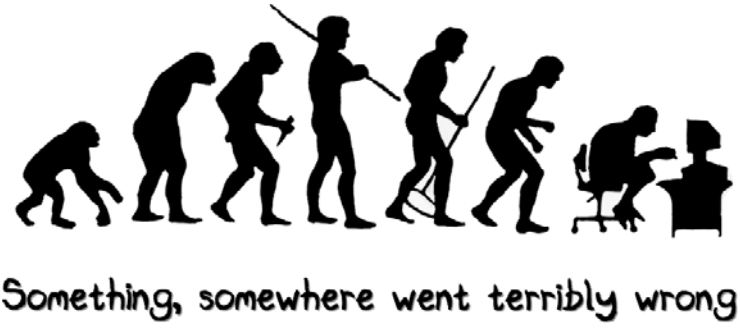

Рис. 3. *Похоже, пользователи изменили свое мнение*

Программное обеспечение может не быть отстоем и не должно быть им, но на практике оно такое, какое есть. Одна из причин этого в том, что программисты, архитекторы и менеджеры, занимающиеся его изготовлением, понимают своих клиентов гораздо хуже, чем следовало бы, и в сравнении с разработчиками в других отраслях. Их продукты отстойны не потому, что разработчики чего-то не умеют, а потому что они не знают, что нужно делать. Они оторваны от своих клиентов («жертв» — говорят некоторые) — тех, кто обеспечивает им зарплату, — и обычно этого не сознают. Часто они решают не те задачи, добавляя новые функции, которые не нужны никому кроме них самих, и нанося ими вред всем пользователям. Если бы они понимали, что за люди их пользователи, то принимали бы другие, лучшие решения.

Например, приложения Microsoft Office, такие как Word и Excel, дают пользователю возможность перетаскивать главное меню (с кнопками File, Edit и Help) с его обычного места в верхней части окна к любому краю экрана или даже оставлять плавающим поверх документа (рис. 4). Я ни разу не встретил ни единого человека и не слышал о таком — подчеркиваю, ни единого, — кто когда-либо воспользовался этой функцией, даже среди моих знакомых разработчиков Office. Вы спросите, что плохого в том, что программа позволяет это делать? Есть несколько отрицательных последствий. Когда я тянусь мышью, чтобы щелкнуть по меню File, я иногда немного промахиваюсь, особенно если перепью кофе, и в результате начинаю перетаскивать панель меню по экрану. Приходится прервать работу, оттащить панель обратно, припарковать на место и полминуты клясть предков тех дебилов, которые написали эту идиотскую «функцию». Ме-

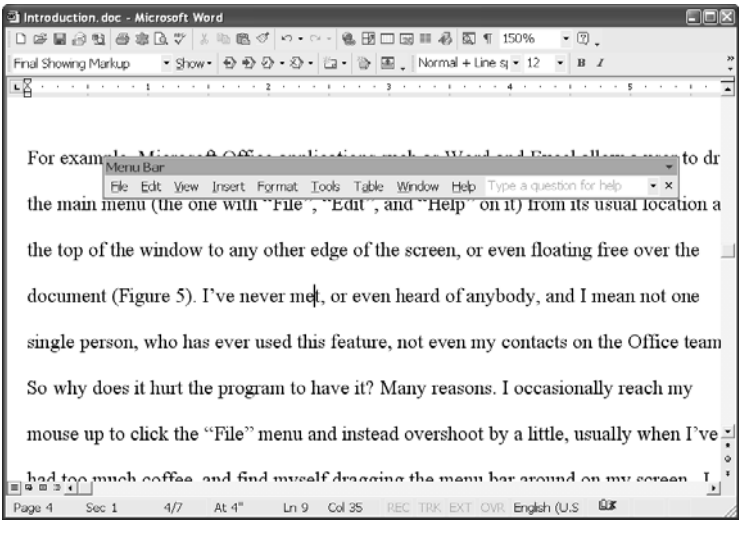

Рис. 4. *Главное меню Microsoft Word, парящее над документом. Сделать-то можно, но кому это надо?*

лочь, но два раза в день по 30 секунд, умноженные на миллиард пользователей, составят около 27 человеческих жизней, выбрасываемых впустую ежедневно.[1] Моя производительность, как и производительность большинства пользователей, станет выше, если эту функцию совсем убрать из программы, чтобы мы не тратили зря время и не теряли концентрации внимания, занимаясь такой чепухой. Кроме того, лишние команды программы («код», как называют их программисты), с помощью которых обеспечивается эта функция, увеличивают вероятность возникновения аварийных ситуаций и уязвимостей в системе защиты, так же как дополнительные движущиеся детали в любом механическом устройстве делают его менее надежным. Жизнь была бы лучше, если бы Microsoft взяла те деньги, которые были потрачены на проектирование, написание, тестирование, отладку, документирование и поддержку этой глупости, и сожгла их, а лучше — отдала мне. Хуже всего то, что такие контрпродуктивные безделушки расходуют ресурсы разработки, которые можно и нужно бы-

[1] Миллиард минут — это 694444 дней, 1901 год или чуть больше 27 раз по 70 лет.

Рис. 5. *Так выглядел Clippy*

ло потратить на то, что действительно важно для большинства пользователей, например, чтобы программа реже загибалась, и, если это случилось, не терялась их работа. Верх расточительности и создания контрпродуктивных функций — это Clippy (рис. 5), говорящая, танцующая и надоедливая скрепка для бумаг в Microsoft Office (добавлена в 1997 году, деактивирована по требованию народа в 2002).[1]

Программное обеспечение и впредь будет отстойным, если мы не примем мер. Наши автомобили стали безопаснее (благодаря надувным подушкам и системам антиблокировки тормозов), надежнее (благодаря лучшему проектированию и сокращению отказов) и удобнее (благодаря плеерам компакт-дисков и подставкам для чашек), только когда этого потребовали покупатели, приобретавшие автомобили, в которых все это было, и игнорировавшие остальные. Мама всегда учила меня, что не стоит осуждать что-либо, если не имеешь представления, как сделать его лучше. У меня много хороших идей, которыми я поделюсь с вами, показывая глупость некоторых проектных решений в современных программах. Эта книга не задумана как сборник советов, но я раскрою все известные мне приемы ослабления самых ужасных результатов плохого проектирования — например, как отключить диалоговое окно подтверждения («Вы уверены?

[1] Я не одинок в своем осуждении этой гнусной горлумоподобной твари. Как написали юристы Далия Литвик (Dahlia Lithwick) и Брандт Гольдштейн (Brandt Goldstein) в книге «Me v. Everybody: Absurd Contracts for an Absurd World» (Я против всех: абсурдные договоры в абсурдном мире), Workman Publishing, 2003: «3. Условие, касающееся маниакальной бумажной скрепки с бровями. Должно быть удалено/отключено/уничтожено все, что дает возможность этой маленькой глупой твари прыгать в правом нижнем углу моего экрана, издавать звуки, которые можно назвать лишь «механическим пуканием», и непрерывно подмечать: «Я вижу, что ты пишешь письмо с требованием выкупа...» или решать, что я хочу все свои документы преобразовать в электронные таблицы, а письма — в нумерованные списки».

Точно уверены? Точно-точно-точно уверены?»), которое появляется при отправке документа в мусорную корзину Windows. Главная задача книги — показать, что мы с вами можем сделать, чтобы наши голоса были услышаны в программной индустрии, и она прекратила выпуск отстоя, желательно скорее. Ведь получилось же с Clippy?

В то же время я должен сообщить вам, что разработка программного обеспечения во многом отличается от разработки физических объектов — чего-то такого, что, скажем, можно уронить себе на ногу. Если, допустим, вы делаете стол из дерева, то свойства материала мешают выбрать одни варианты конструкции и вынуждают выбирать другие. Например, понятно, что нельзя паять, но можно крепить винтами, и толщина должна быть достаточно большой, чтобы стул не прогибался. Программное же обеспечение, напротив, предполагает очень мало внутренних ограничений. Гибкость материала здесь чрезвычайно велика. Как писал Фредерик Брукс в классической книге о разработке программного обеспечения «Мифический человеко-месяц» (Addison Wesley, 1995), которую я бы предложил ему переименовать в «Мистическую гико-неделю» соответственно сокращению сроков проектов в нынешние времена, программисты работают «почти исключительно с чистой мыслью». Я всегда говорил, что программирование — это попытка затолкать дым в бутылку. Поэтому я объясню вам, какие вещи действительно трудно заставить делать программу по самой природе (например, успешно переносить внезапное выключение питания или быть совместимой с прежними версиями[1]), а какие обусловлены просто безмозглым проектным решением тупого разработчика, который ни черта не соображает.

Я преподаю разработку программного обеспечения на факультете непрерывного образования Гарвардского университета и в различных компаниях во всех концах света. Я написал девять книг для программистов и руководителей программных разработок, а также много статей для журналов и информационных бюллетеней. Несмотря на эти недостатки, я обещаю, что мои тексты не будут скучными. Я старательно стремился избегать технического жаргона. Например, вы не встретите в этой книге слова

[1] В любимой шутке программистов говорится: «Как Творцу удалось создать мир всего за шесть дней? У него не было предыдущих инсталляций, совместимость с которыми необходимо было обеспечить». Мой редактор заметил, что Он также сэкономил на документации.

«гигабайт»[1], вместо которого я стану употреблять слова «четверть пространства жесткого диска», что на момент написания было примерно равноценно.

В сравнении с прочими моими книгами эту было удивительно легко писать. Покойный спортивный комментатор Ред Смит (Red Smith, 1906–1982) любил говорить: «Писать легко. Просто вскрываешь себе вены и пишешь». Надеюсь, вы уже заметили, что я весьма эмоционально отношусь к некоторым затрагиваемым здесь темам. Когда я заканчиваю свои шумные речи («Windows XP — приличный для своей цены продукт. Windows 98 — это кусок сами знаете чего, не стоящий затрат на то, чтобы его выкинуть к черту»), мои слушатели часто говорят «Платт, хватит ходить вокруг да около и скажи, что ты думаешь на самом деле». На что я отвечаю: «Если вы меня обвиняете в том, что я говорю о них то, что вижу, то я признаю себя виновным по этому пункту». Эта книга не просто попытка объяснить вам положение; это мое обращение к небесам в желании, чтобы здравомыслие возобладало в этой безумной профессии. Надеюсь, что, прочтя эту книгу, вы присоедините свой голос к моему.

[1] Ну да, только один раз и только здесь.

ГЛАВА ПЕРВАЯ

Кого считать идиотами?

«Ее никто не купит, — усмехнулся я, увидев в магазине название книги. — Кто на глазах у всех станет покупать книгу, которая возвещает всему миру, что он идиот? Это все равно, что покупать презервативы, украшенные надписью „для самых маленьких“».

Но все мы теперь знаем, что из этого вышло. «DOS для чайников» и парная к ней «Windows для чайников» стали абсолютными бестселлерами книг по компьютерной тематике. Идея вышла далеко за пределы компьютеров, и появились такие книги, как «Вино для чайников», «Аквариумы с соленой водой для чайников» и «Рак груди для чайников». В этой серии было продано более 100 млн. экземпляров, если верить книге «Как опубликовать свою книгу для чайников», которую я купил, чтобы найти издателя для того труда, который вы сейчас держите в руках.[1]

Компьютеры заставляют людей чувствовать себя тупыми. Грамотные, образованные люди не могут заставить этот проклятый ящик делать то, что им нужно, и вместо того чтобы двинуться на Microsoft с факелами и вилами и вздернуть там чучело Билла Гейтса, они клянут себя, говоря: да,

---

[1] Результаты бывают неприятными. В октябре 2003 года комиссия по безопасности потребительских товаров сняла с продажи «Изготовление свечей и мыла для чайников», поскольку некорректные инструкции по смешиванию некоторых химикалий были чреваты получением ожогов. Не знаю, как следует в этой связи относиться к книге о раке груди или другой из той же категории — «Рак простаты для полных идиотов».

наверное, я тупой. В обществе, в котором никто никогда не виноват, в котором человек, проливший на себя кофе, подает иск к ресторану, где он это сделал, заставить пользователей винить себя за что-то — потрясающее достижение, хотя, возможно, не это было главной целью производителя. Почему программисты создают приложения, которые заставляют людей испытывать такие чувства, и почему люди смиренно терпят такие оскорбления от своих компьютеров?

Что было раньше

Разработчиков самых первых компьютерных программ не заботило, легко ли пользоваться их продуктами. Решать стоявшие перед ними задачи, например заставить принтер правильно выводить слова на бумагу, было столь сложно, что не оставалось ни времени, ни денег, чтобы облегчить жизнь пользователя. Время работы компьютера стоило чрезвычайно дорого — гораздо дороже, чем время пользователей. Заставлять человека учить сложные команды вместо того, чтобы потребовать от компьютера выводить их список в виде меню, было оправданно экономически. Сейчас соотношение цен стало обратным, но каждый работник компьютерной отрасли, кому сейчас за 30, вырос в тех старых условиях. Это, несомненно, отразилось на нашем сегодняшнем мышлении, как бы мы ни стремились расстаться с прошлым. Вспомните своих пожилых родственников, выросших во время Великой депрессии 1930-х годов: они даже сегодня не понимают, как можно выбрасывать носок, в котором всего одна дырка.

Первые пользователи компьютеров были готовы к тому, что их ждут трудности, не уступающие проблемам водителей авто в начале XX века, и их опасения обычно оправдывались. Почти все пользователи сами были программистами. Они редко испытывали потребность или даже желание облегчить положение. Мы воспринимали сложности — нехватку компьютерного времени, непонятность команд, низкое качество документации — так же, как те пионеры-мотористы воспринимали двигатели, заводящиеся рукояткой или постоянные проколы шин. Все были в одинаковых условиях. Мы были счастливы одной возможностью выполнять сложные вычислительные задачи (составление таблиц по результатам переписи населения, взлом кодов противника), так же как они были счастливы, что избавились от необходимости каждый день выгребать из конюшни навоз. Нам нравилось возиться со своими программами, находя им такое применение,

которое не предполагалось их разработчиками, так же как первым механикам нравилось копаться в своих двигателях. Генри Форд рассмеялся бы в лицо тому, кто сказал бы, что в его модели «Т» не хватает держателя для чашки.

В те времена бытовало представление, что облегчать работу с программами не следует. Если программу трудно написать, значит, и пользоваться ею должно быть тяжело, чтобы плоды программистских трудов могли пожинать только те, кто своими интеллектуальными усилиями доказал на это право. Даже сегодня я с удивительной нежностью вспоминаю гордость, которую испытал, когда сделал открытие, что команда для печати документа на той первой солидной вычислительной системе, с которой я работал (в 1975 году, на первом курсе колледжа), называется не Print или P, а Q, потому что вы ставите документ в queue — очередь для печати. Я узнал волшебное слово. Я входил в круг избранных. Я был Умен!

Но когда аппаратная часть подешевела и компьютеры переместились из охлаждаемых кондиционерами застекленных зон, где над ними колдовали верховные жрецы, на верстаки умельцев-любителей, а после и на рабочие столы обычных служащих и в дома обычных людей, они обязаны были стать проще в обращении. Поэтому разработчикам приложений пришлось вкладывать время и деньги в разработку таких программ, с которыми пользователи действительно могли бы работать. Почему же результаты оказались такими жалкими?

Почему и по сей день программы — отстой

Та часть компьютерной программы, которая взаимодействует с человеком-пользователем, то есть получает от него команды и входные данные, выводит сообщения и результаты работы, называется **интерфейсом пользователя**. Разработка интерфейса пользователя, так же как и многие другие области вычислительной науки, — искусство, требующее специальных умений и знаний, о котором большинство программистов не имеет никакого понятия. Программистами они стали потому, что научились общаться с микропроцессором — кремниевой микросхемой, составляющей основу вычислительной машины. Но интерфейс пользователя по определению предназначен для общения с совершенно другим программно-аппаратным устройством — живым человеком. Нет ничего удивительного в том, что искусство беседы с логичным, безошибочным и глупым чипом

совершенно отлично от искусства беседы с нелогичным, склонным ошибаться и разумным человеком. Однако считается, что тот, кто преуспел в первом искусстве, автоматически должен справиться со вторым. Обычно это не так, и сам программист этого почти никогда не сознает. Вот поэтому интерфейсы пользователя, созданные программистами, обычно отвратительны — во всяком случае с точки зрения того бедолаги, который мучается с этим хламом.

Почему так происходит? Ведь раз программисты умеют программировать, значит, определенный уровень интеллекта у них есть. В большинстве своем они неплохо ладят с этой кремниевой микросхемой, иначе бы их довольно быстро уволили и им пришлось бы подбирать себе другую профессию, в которой они могли оказаться полезнее обществу, например заняться кровельными работами. Почему же, принимаясь за разработку интерфейса пользователя, они превращаются в слабоумных идиотов? Причина проста, и она всегда лежит в основе любых мыслимых случаев взаимного непонимания: они не знают своих пользователей.

Каждому программисту кажется, что ему достоверно известно, чего хотят пользователи. В конце концов, он же ежедневно с утра до вечера работает с компьютером и не может этого не знать. Он говорит себе: «Если я сделаю такой интерфейс пользователя, который нравится мне, то и пользователи его полюбят». *Ничуть не бывало!* Если только программист не пишет свои программы для таких же прожженных технарей, как он сам, пользователям это не понравится. Своим ученикам, которых я учу программировать, я советую зарубить на носу — наряду с фразами типа «мусор на входе, мусор на выходе» и «всегда снимай колоду» — Первый Закон Платта, Единственный Закон Проектирования Пользовательского Интерфейса:

Познай своего пользователя, ибо он не такой, как ты

В качестве простейшего примера возьмем программу управления личными финансами типа Quicken или Microsoft Money. Пользователь имеет с ними дело по несколько часов раза два в месяц. Естественно, ему трудно вспомнить, как управляться с этой программой, — в отличие от той, с которой он работает ежедневно. Поэтому подсказки и помощь должны быть детальными — такими, которые покажутся назойливыми и раздражающими человеку, не вылезающему из этой программы (например, программисту). Программист не в состоянии поставить себя на место этого

пользователя. Программист очень хорошо знает свою программу и не может себе представить, что не все такие же, как он.

В своем заблуждении, что все люди похожи на них, программисты совершают две главные ошибки, проектируя интерфейс пользователя. Они придают большее значение степени контроля, а не легкости использования, и стремятся к тому, чтобы можно было делать сложные вещи, вместо того чтобы сделать простые вещи простыми. И кроме того, они полагают, что пользователей интересует внутреннее устройство их программы, что не соответствует действительности. Мне приходилось совершать обе ошибки, и я сожалею о заблуждениях своей молодости.

Контроль или легкость эксплуатации?

Каждый раз, читая лекции в какой-нибудь компании, я спрашиваю у слушателей, многие ли из них ездят на машинах с ручной коробкой передач (как я сам). Обычно руки поднимают около половины. Тогда я спрашиваю, многие ли *приехали бы* на машине с ручной передачей, если бы им это позволили жены, или они поехали на машине с автоматической коробкой, потому что превращаются в брюзгливых старикашек вроде меня. Обычно руки поднимает половина из оставшихся.[1] «Теперь согласитесь, — говорю я, — что ручная передача сложнее в освоении и эксплуатации, чем автоматическая, но обеспечивает лучшую управляемость?» Они чувствуют, что здесь какая-то ловушка, но обычно еще не могут ее распознать, поэтому соглашаются, но неохотно. «А теперь скажите, какой, по-вашему, процент машин с ручной передачей продается в США?» Они неловко ежатся и обычно говорят что-то вроде «мало, процентов 30». Размечтались. По разным оценкам от 10 до 14 процентов. Пусть будет 12 с половиной, для простоты.

Этот опрос показывает, что шесть из восьми программистов ценят незначительное преимущество в степени контроля и эффективности настолько высоко, что, потратив 25 или более тысяч долларов на своего железного коня, они готовы на постоянную дополнительную работу в тече-

[1] Сделайте такую проверку в своей компании. Благодаря ей вы, возможно, узнаете нечто новое о своих пользователях. Потом зайдите на сайт этой книги (*www.whysoftwaresucks.com*) и сообщите мне полученные результаты. Спасибо.

ние всего срока службы автомобиля, чтобы этим преимуществом пользоваться. И лишь один из восьми человек среди всего населения принимает такое же решение при тех же условиях выбора. А на самом деле еще меньше, потому что все эти шесть программистов также входят в эту одну восьмую. Процент нормальных людей, стремящихся выполнять лишнюю работу, близок к нулю. Программисты ценят контроль. Пользователи ценят легкость эксплуатации. Ваш пользователь не такой, как вы.

Вот пример, показывающий, как можно все испортить. Когда-то справочная служба AT&T была простой и удобной. Вы спрашивали чей-нибудь номер, и автоматический голос отвечал «Нужный вам номер: 555-1212. Пожалуйста, запишите его». Если вы не положили трубку, номер произносился еще раз, и можно было проверить, правильно ли вы записали его. Просто. Легко. Невозможно ошибиться. Отлично. Затем в AT&T ввели возможность автоматического набора этого номера. Голос произносил: «Нужный вам номер 555-1212 можно автоматически набрать за дополнительную плату 50 центов. Нажмите 1, если вы согласны, и 2 в противном случае». Простота сохранилась, но появилась новая более мощная функция для тех, кто готов был за нее заплатить. Тот, кому она была не нужна, мог просто положить трубку. Потом какому-то айди-ойту [*sic*, см. примечание[1]] пришла в голову абсолютно дикая мысль. Когда я в прошлый раз хотел воспользоваться справочной AT&T, мне было сообщено: «Нужный вам номер можно автоматически набрать за дополнительную плату 50 центов. Нажмите 1, если вы согласны, или 2 в противном случае». Служба не называет номер, пока я не нажму одну из кнопок. Мне пришлось отнять трубку от уха, найти глазами клавиатуру (а лет после 45 это уже труднее), положить карандаш, который я держал в руке, чтобы

[1] Происхождение слова связано с курсом, который я однажды читал. Некий слушатель слабо успевал, потому что не слишком утруждал себя, и взял привычку писать мне длинные и эмоциональные электронные письма, жалуясь на несправедливость. Конечно, он был очень расстроен. С другой стороны, трудно воспринимать всерьез того, кто якобы учится на последнем курсе престижного университета и не научился правильно писать слово «идиот». Обычно он писал мне: «Платт, вы айдиойт. И тот, кто ставит оценки, — айдиойт, и тот, кто мне посоветовал этот курс, — айдиойт, и я — айдиойт, потому что послушал его». С тех пор мы с коллегами пользуемся словом *idoit* (произносится «айдиойт», или «идуа», если вы француз) для обозначения людей настолько невежественных, что они не могут даже правильно написать «идиот».

записать номер, нажать нужную кнопку, снова взять карандаш и приложить трубку обратно к уху. Только после этого мне сообщили, что нужный номер — 555-1212. Сложная и мощная операция возможна, но простая операция перестала быть простой. Разработчик этой системы, очевидно, ценит контроль выше простоты, но гарантирую, что пользователи придерживаются другой точки зрения. Того, кто придумал такую операцию, следовало бы заставить проделывать ее 500 раз в день. К концу недели он бы застрелился.

Мой оператор сотовой связи Verizon, напротив, поднял простоту пользования на новую высоту. В Verizon сообразили, что почти все, кто обращается в справочную службу, хотят кому-то срочно позвонить, так почему бы не помочь? Когда я звоню в справочную службу с сотового телефона, автоматический голос произносит: «Номер 555-1212. Соединяю вас». Все происходит автоматически, без каких-либо действий или даже мыслей с моей стороны. Новый номер сохраняется на моем телефоне в списке последних набранных, поэтому я могу при желании поместить его в свой справочник. Те немногие, кто собирался только записать номер, могут просто повесить трубку, что они и так сделали бы. Простые вещи оказываются простыми. Сложные и мощные — тоже простыми. Такая схема настолько же хороша, насколько плоха схема AT&T.[1]

Меня не интересует, как работает ваша программа

Вторая ошибка, которую допускают программисты при разработке пользовательского интерфейса, заключается в их стремлении заставить пользователя разобраться во внутренней логике своей программы. Вместо того чтобы приспособить интерфейс к способу мышления пользователя, программист хочет заставить его приспосабливаться к своему способу мышления. Более того, программист считает такой подход правильным. «Но ведь так работает моя программа», — говорит он, недоумевая, как у кого-то могла возникнуть мысль сделать иной интерфейс.

---

[1] Я только что прочел, что Verizon собирается передавать на ваш телефон маршрут проезда к тому месту, телефонный номер которого вы запросили, установив ваше положение с помощью встроенного в телефон чипа GPS. Очень рассчитываю, что они ничего при этом не усложнят и не испортят.

Приведу пример, показывающий, что я имею в виду. Откройте Блокнот Windows или любую другую программу-редактор и наберите с клавиатуры произвольный текст. Теперь выберите в главном меню пункты File→Exit или щелкните по крестику в правом верхнем углу. Окно сообщения, которое вы увидите, показано на рис. 1.1.

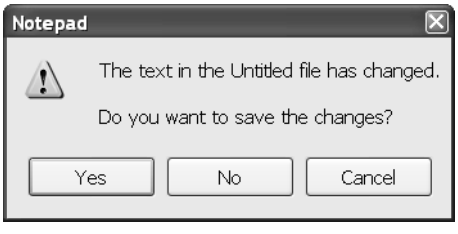

Рис. 1.1. *Блокнот спрашивает у пользователя, нужно ли сохранять изменения*

О чем она спрашивает, эта программа? Говорит, что какой-то файл изменился, но я нигде не видел никаких файлов. Какие, к черту, изменения, которые нужно сохранять?

Дело в том, что Блокнот обычно применяется для редактирования документов («файлов» на техническом жаргоне), находящихся на жестком диске. Когда вы открываете документ, Блокнот копирует его с диска в память компьютера. Когда вы добавляете или удаляете текст с клавиатуры, программа изменяет содержимое этой копии в памяти. (В данном примере мы не открывали существующий документ, а программа создала в памяти новый документ, назвав его «Безымянный».) После того как вы завершите работу с документом, программа должна записать его из памяти обратно на диск, и эта операция называется «сохранить файл». Если этого не сделать, вся ваша работа пропадет, и вы очень рассердитесь.

Программист написал программу, которая работает подобным образом (копирует документ с диска в память, изменяет копию, находящуюся в памяти и записывает ее обратно на диск), потому что для него это было проще всего. И он выбрал совсем неплохой способ. Чтение или запись символов с диска (вращающихся металлических пластинок) происходит в тысячи раз медленнее, чем те же действия с памятью (электроны движутся со скоростью света), поэтому возможно, что такой способ организации внутренней логики программы действительно самый лучший.

Но интерфейс пользователя, который сделал этот программист, просто отображает внутреннюю логику программы. Что в этом плохого? Автор программы заставляет вас разбираться в том, как она работает. А вам не нужно знать или думать о том, как устроена программа внутри, чтобы успешно пользоваться ею, так же как для управления автомобилем вам не нужно знать или думать о том, впрыскивается в двигатель топливо или поступает из карбюратора.

Обычно ваши мысли идут не таким путем, какой предлагает эта программа. Большинство людей представляет себе редактирование документа в компьютере аналогичным работе с карандашом и бумагой (вы еще помните, что это такое?). Вы делаете пометки карандашом, и они остаются на бумаге. То, что вам не нужно, вы стираете резинкой. Если документ вам больше не нужен, вы комкаете лист и выбрасываете его. Бумага хранит всю вашу работу, если только вы не потратите дополнительную энергию, чтобы уничтожить ее. Но Блокнот предлагает вам другое. Каждый новичок в компьютерах на этом попадается — говорит «нет», и Блокнот выкидывает всю вашу работу — хорошо, если ее немного. В конечном счете пользователь привыкает думать, как компьютерная программа, точнее, как программист, который написал эту чепуху. Алан Купер, гуру в области проектирования интерфейсов пользователя, определяет «грамотного пользователя компьютера» как того, кто «набил себе столько шишек, что у него одеревенела шкура и он больше не чувствует боли».

Вопрос и ответ на него были бы намного понятнее, если бы текст окна диалога гласил: «Выбросить все, что ты только что сделал?» Это абсолютно тот же самый вопрос, но заданный с точки зрения пользователя, а не программиста. Но программист думает только о том, что делает программа, то есть о записи на диск, и спрашивает, нужна ли запись. Автор требует, чтобы вы встали на его место, даже не попытавшись встать на ваше. Иначе он сформулировал бы вопрос по-другому. Иначе он увидел бы нелепость своего вопроса и создал более удачный интерфейс пользователя, даже если бы программа, которая за ним скрывается, продолжала работать по-прежнему.

Программа управления личными финансами Microsoft Money устроена правильно. Тот, кто ее проектировал, понимает, что мысленной моделью для пользователя служит чековая книжка, поэтому он и показал на экране опись чеков (рис. 1.2). Пользователь-новичок чувствует себя здесь привычно и удобно (ну, относительно). Чек, с которым вы работаете в дан-

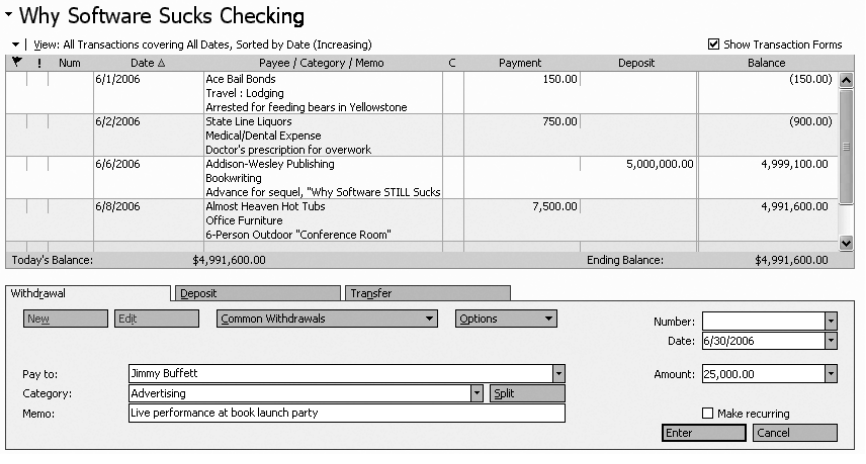

Рис. 1.2. *Интерфейс пользователя Microsoft Money, похожий на настоящую чековую книжку*

ный момент, выделен другим цветом. Вы вводите в чек нужные данные и нажимаете Enter. Чек перемещается вверх, цвет его становится таким же, как у остальных, а в рабочей области появляется новый пустой чек. Если включен звук, раздается «динь-динь», как в кассовом аппарате.[1] Программа не спрашивает у вас, нужно ли сохранить чек. Само нажатие клавиши Enter говорит программе, что вы хотите сохранить информацию. Если вы потом передумаете и захотите поменять дату чека или совсем удалить его, вам придется щелкнуть по этому чеку и ввести новые данные. Спрашивается, когда эта программа читает данные с диска и пишет их обратно? Не знаю и знать не хочу. И вам это знать ни к чему. Интерфейс пользователя этой программы следует вашей мысленной модели, а не заставляет вас что-то изучать и разбираться с внутренними конструктивными решениями программиста.

Такой тип проектирования интерфейса пользователя гораздо лучше. Я, будучи пользователем, не хочу думать о самой программе. Я хочу ду-

[1] Этот звук, конечно, анахронизм. Где вы сейчас услышите такие позвякивания у кассового аппарата? Современные кассы попискивают и шумят, как компьютеры, каковыми они и являются. Но этот звук сохранился, как сохранилось выражение «повесить телефонную трубку», хотя мы всего лишь нажимаем кнопку.

мать о работе, которую делает для меня программа, например достаточно ли у меня денег, чтобы оплатить этот счет? Еще один гуру в области дизайна интерфейса Дональд Норман (Donald Norman) очень хорошо выразил это представление в названии одной из своих книг — «The Invisible Computer» (Невидимый компьютер), MIT Press, 1999. В идеале я вообще не должен думать о программе.

Это одна из важных причин, по которым с программами трудно работать, и они заставляют вас чувствовать себя дураком. Вас принуждают думать по-программистски, даже если вы не программист и не собираетесь им быть. А вам это и ни к чему. Не надо быть механиком, чтобы водить машину, врачом, чтобы принять аспирин, и мясником, чтобы поджарить гамбургер. Вы платите за этот продукт свои деньги, заработанные тяжелым трудом. Это программист должен подстраиваться под вас, а не наоборот.

Плохая функция и хорошая функция

А вот еще один пример того, как программисты уродуют интерфейс и заставляют пользователей чувствовать себя тупыми. Выберите на рабочем столе Windows любой документ. Теперь нажмите клавишу Delete. Если только вы не сумели ранее отключить эту функцию, перед вами появится окно подтверждения, показанное на рис. 1.3, спрашивающее, действительно ли вы хотите удалить этот файл.

Был хоть один раз, когда вы сказали «ой, я не собирался это делать, спасибо за предупреждение» и нажали No? Может быть, вы видели, как это делает кто-то другой, или слышали про такой случай? Лично я — нет. Подтверждениями так часто злоупотребляют, что по иронии судьбы они стали совершенно бесполезными. Это окно постоянно кричит «волк», как пастушок в басне Эзопа, поэтому никто не обращает на него внимания,

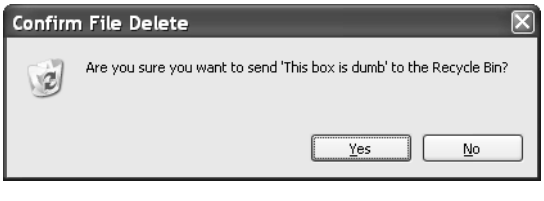

Рис. 1.3. *Бессмысленное окно подтверждения от мусорной корзины Windows*

даже когда предупреждение относится к файлу, который вы действительно не собирались удалять. Вы так часто с этим встречались, что уже не обращаете внимания. Действуя на автопилоте, вы автоматически нажимаете кнопку Yes. Безопасности вы не получаете. Никакой. К счастью, именно это диалоговое окно можно отключить.[1] Есть масса других, от которых вам избавиться не удастся, хотя их быть не должно. Совсем. Нигде. Никогда.

В жизни вы совершаете разные операции, и при этом у вас не требуют подтверждения. Когда вы поворачиваете ключ зажигания, ваша машина не спрашивает у вас, действительно ли вы хотите завести двигатель. В супермаркете служащий не спрашивает у вас, действительно ли вы хотите купить эти товары, когда вы выкладываете их на ленту у кассового аппарата. Вспомните, сколько еще книг вы купили на Amazon.com после того, как обнаружили их фирменную функцию «заказ одним щелчком».[2]

Зачем программисты все время просят подтверждений? Они делают это потому, что им кажется, будто их пользователи запутались и не понимают, каковы будут последствия того, о чем они только что попросили программу. Не исключено, что так оно и есть — с учетом ужасного качества остальных частей интерфейса пользователя. Но запрос подтверждения не решает этой проблемы. Если пользователь был сбит с толку, когда изначально дал ту команду, которая вывела диалоговое окно подтверждения, то после его появления он запутается еще больше. Поскольку программа показывает нежелание выполнять то, что ей сказано, он начинает думать, что сделал какую-то ошибку. Применение окон запроса подтверждения избавляет программистов от того, чтобы а) четко объяснить пользователю последствия его действий, чтобы он не попытался сделать то, что ему не нужно, и б) предоставить возможность восстановить ситуацию, если пользователь действительно совершит то, о чем позднее пожалеет.

---

[1] Щелкните правой кнопкой по Корзине, выберите Свойства во всплывающем меню и сбросьте флажок Показывать запрос подтверждения.

[2] Мало кто знает, что в первом прототипе этой функции программисты Amazon выводили диалоговое окно подтверждения с вопросом «Вы действительно хотите заказать этот товар одним щелчком?», когда пользователь щелкал по кнопке «заказ в 1 щелчок», тем самым превращая его в 2-щелчковый процесс. Они бились за сохранение этой функции, пока не последовало прямое указание президента Amazon Джеффа Безоса убрать ее и сделать действительно процедуру одного щелчка.

Но что если пользователь действительно совершает ошибку? Если, например, вы положите на транспортер у кассы фонарик и комплект батареек, которые для него не годятся, разве внимательный служащий не поинтересуется, действительно ли они вам нужны? Разве не должен хороший интерфейс пользователя оберегать нас от таких ошибок? Конечно, должен, и прелесть компьютерных программ в том, что они в состоянии это делать. Но это достигается не с помощью задаваемых каждый раз глупых вопросов «а вы уверены, что действительно хотите сделать эту ерунду, о которой только что меня попросили?» В хорошем интерфейсе пользователя такой проблемы вообще не должно возникать. Например, на вебстранице, где продается фонарик, может быть флажок «вместе с батарейками». По умолчанию он должен быть установлен, потому что без батареек фонарик не будет работать. Если у покупателя уже есть достаточное количество батареек такого размера, он может сбросить флажок. А еще лучше, если фонарик будет упакован уже с батарейками внутри, чтобы быть готовым к работе в тот момент, когда вы его достанете из упаковки, не задумываясь о батарейках. Толковый разработчик интерфейса пользователя подумает об этом еще до начала программирования. Если программисту кажется, что необходимо показать окно подтверждения, то я вам гарантирую, что он напортачил где-то в интерфейсе, иначе оно не понадобилось бы. Он, наверное, и не пытался обойтись без него, ему, наверное, и в голову не пришло такое. Окно подтверждения — это подпорка для ленивого или несведущего программиста, расплачиваются за которую все пользователи. К тому же она не действует.

Но разве не было бы желательно подтверждать намерение совершить некое деструктивное действие типа удаления файла? Нет. Еще одна из причин, по которым не требуется подтверждать намерение завести двигатель или купить товар в супермаркете, состоит в том, что эти операции легко и просто отменить, если вы решите, что сделали ошибку. Вы просто выключаете зажигание или возвращаете ненужный товар. Компьютерные программы могут легко и просто делать копии документов или участков памяти. Благодаря этому программисты могут предоставлять пользователям возможность отменять выполненные действия. Функция `Undo` (Отмена) — одна из самых полезных в арсенале разработчика интерфейса пользователя. По моим представлениям, это самое большое отдельное достижение в проектировании интерфейса пользователя со времени появления мыши.

Когда вы нажимаете кнопку Yes в окне подтверждения на рис. 1.3 (или если вы совсем отключили это окно), Windows не стирает документ в компьютере, а перемещает его в другую область диска под названием Корзина (Recycle Bin), аналогичную известному мусорному ящику на Макинтошах. Если после этого вы передумаете и пожелаете получить документ обратно, то можете достать его из Корзины, если только перед тем не очистили ее. Как правило, вам действительно нужно убрать файл. Гораздо эффективнее исправлять относительно небольшое количество ошибок, которые действительно случаются (например, можно выбрать для удаления не тот файл, неточно переместив указатель мыши, — со мной такое бывает), чем пытаться предотвратить такие ошибки, раздражая пользователей показом окна подтверждения при каждом удалении, особенно если на него привычно не обращают внимания. Ложка лекарства не стоит бочки профилактики.

Функция «отменить» может применяться не только к операциям с файлами, но и внутри приложений. Обычно она располагается в меню Edit (Правка) вместе с парной к ней Redo (Повторить), которая, разумеется, отменяет отмену. У меня обычно и пяти минут не проходит, чтобы я не отменил какое-нибудь действие: либо напечатаешь дурацкую фразу, либо текст переместишь не туда, куда нужно. Программисты, которые реализуют такую функцию, — настоящие друзья пользователя. При случае я всегда угощаю их пивом, и вам советую поступать так же. Заставить эту функцию действовать так, чтобы у пользователя не возникало никаких проблем при ее вызове, неимоверно трудно (как известно, «чтобы было легко, нужно немало потрудиться»): всего лишь нажать Ctrl-Z (мизинцем и средним пальцем левой руки) — и вот оно вернулось, как будто ничего не произошло. Программа должна просить подтвердить только те операции, которые она не сможет отменить. А отменить она должна уметь все.

Подлинная прелесть Undo в том, что она позволяет исследовать программу. Иногда бывает непросто понять, как действует незнакомая вам программа, ориентируясь только по коротким надписям в пунктах меню и миниатюрным картинкам на кнопках в панели инструментов. Но команда Undo есть, поэтому можно поэкспериментировать, проверяя разные варианты, и не бояться что-либо испортить, чего нельзя будет исправить несколькими нажатиями клавиш. Можно подвигать абзац и оценить, не будет ли смотреться лучше в другом месте, а если нет, то вернуть его обратно. Программисты часто относятся к вводу неправильных данных как к действиям идиота, не удосужившегося сесть и прочесть руководство.

И они ошибаются. Это основной метод, используемый человеческим родом при обучении. Приложение с функцией Undo становится обучающим, а не устрашающим. Оно признает и еще больше укрепляет человеческую природу пользователя. Неспособность реализовать эту функцию надлежащим образом — смертный грех.

Прекращение работы и демонстрация идиотизма

Если Undo реализована правильно, деструктивная операция в системе остается одна, а именно очистка Корзины. Кое-кто скажет, что уж у этой-то операции должно быть диалоговое окно подтверждения, и оно действительно есть. Но даже тут окно подтверждения служит лишь защитой против другого скверного проектного решения — расположения пункта контекстного меню Explore (Проводник) рядом с пунктом Empty Recycle Bin (Очистить Корзину). Одно неточное движение мыши, проскочившей три строчки меню вместо двух, и вы получаете одно вместо другого. Это плохо. Если это единственное деструктивное действие в системе, то очистка Корзины должна осуществляться особым образом, больше ни для чего не используемым, например одновременным нажатием двух кнопок мыши («взятием аккорда») или щелчком при одновременном нажатии какой-либо клавиши. А еще лучше было бы очищать корзину автоматически, например удаляя из нее файлы по истечении определенного промежутка времени, величину которого можно было бы задавать, начиная, скажем, с месяца, чтобы пореже приходилось опустошать корзину вручную. Вот бы дома иметь такие мусорные бачки! Окна с запросами подтверждения не должны появляться нигде и ни при каких обстоятельствах. Программист, который их создает, не справляется со своими обязанностями и не должен заниматься разработкой интерфейсов пользователей.

Худшие свои черты интерфейсы, разработанные программистами, проявляют при показе пользователю сообщений об ошибках. Как раз сегодня я, вместо того чтобы работать над этой главой, залез на домашнюю страницу *CNN.com* и захотел сохранить ее у себя на диске. Я выбрал в меню веб-броузера File→Save. Появилось диалоговое окно, сообщавшее о ходе операции сохранения — 5% сохранено, 15% и т. д. вплоть до 99%. Затем это окно исчезло и выскочило то, которое показано на рис. 1.4.

Это окно — творение полного идиота. Почему нельзя было сохранить веб-страницу, и можно ли исправить ситуацию? Была ли страница каким-

Рис. 1.4. *Очень глупое диалоговое окно*

то образом защищена от копирования, как пытаются иногда делать на сайтах профессиональных художников? Или сервер оказался недоступен? Почему индикатор выполнения дошел до 99 процентов, если операция была обречена? Где те 99 процентов страницы, которые были сохранены, как о том сообщила программа? Пусть это не вся страница, но я предпочел бы 99 процентов, нежели ничего. Куда она исчезла? В окне сказано, что страницу нельзя сохранить по выбранному адресу; надо ли это понимать так, что по другому адресу ее можно было сохранить? Если да, то где и как это узнать? Если нет, то причем тут адрес? Броузер уже сумел удовлетворительно показать мне эту страницу, поэтому я захотел ее сохранить. Почему он не может сохранить те данные, которые фактически показывает? В окне ничего не сказано относительно того, как узнать, в чем проблема и где найти дополнительные сведения. А в качестве ответа предлагается только кнопка OK. Нет, для меня совсем не «OK», если операция не выполнена и программа не может объяснить почему. Даже заголовок окна «Ошибка сохранения веб-страницы» вводит в заблуждение. Я не совершал ошибок. Я сделал то, что программа мне позволяла. Это программа сделала одну ошибку, не сумев сохранить мою страницу, а потом другую, не сумев объяснить причину. Создать такую неразбериху с помощью полутора десятков слов — верх идиотизма, с которым мне приходилось встречаться.

Алан Купер, уже упомянутый мной гуру проектирования интерфейса пользователя, описывает подобные ситуации как «прекращение работы и демонстрация идиотизма». Если сохранить страницу нельзя, броузер должен был это знать и предупредить меня, дав какое-то объяснение, — желательно не в виде очередного дурацкого окна. Может быть, при показе этой страницы нужно сделать неактивным пункт меню «сохранить» или изменить текст на «сохранение невозможно — страница защищена», чтобы я знал, в чем дело. Если нельзя сохранить страницу целиком, программа

должна сохранить то, что может, и сообщить мне о пропущенном, — опять же не вынуждая меня щелкать по очередному дурацкому окну. Возможно, на сохраненной странице следует поместить какие-то маркеры в тех областях, которые сохранить не удалось, и показывать красные крестики, как это делает броузер для ненайденных элементов страницы.

Проведя некоторые изыскания, я смог восстановить картину происшедшего. В настройках моего броузера я заблокировал некоторые виды раздражавшего меня контента. Броузер показывает их как пустые области, что нравится мне гораздо больше, чем дурацкие пляшущие рекламные объявления, которые там должны были быть. (Об идиотизме пляшущей рекламы мы поговорим в другой главе.) Когда программный код, который должен был сохранить страницу, добрался до этих элементов страницы и обнаружил, что они заблокированы, он не пропустил их, как это сделала часть программы, ответственная за отображение страницы. Код поперхнулся в этом месте, прервал всю процедуру вместо сохранения того, что можно было, и прекратил работу, продемонстрировав описанный выше идиотизм.

Что нужно, чтобы не чувствовать себя идиотом, когда тебе показывают такое непонятное окно? Нужно знать, что здесь не ваша вина, а неспособность программиста справиться со своими обязанностями; что пользователь не обязан понимать такие глупые сообщения. Полезно также представить себе, как, схватив этого программиста за горло, вы наносите ему коленом удары в пах. Здесь отсылаю вас к советам в конце этой главы и в заключительной главе этой книги.

Испытание на подопытных животных

Программист не выпустит продукт, не протестировав его внутренней работы (OK, *не должен выпустить*). Почему же тогда он считает, что можно обойтись без тестирования интерфейса пользователя и не проверять, смогут ли пользователи с ним работать? Потому что ему самому продукт нравится, и он сам может с ним работать, значит, и у других получится. Как мы уже видели, такое подсознательное предположение почти всегда ошибочно. Если пользователь не может разобраться, как работать с программой, компьютер превращается в очень дорогое пресс-папье. Проверка интерфейса пользователя, которая называется **тестированием юзабилити**, — процедура сложная и дорогая, но необходимая.

Недостаточно просто отдать пользователям программу и потом поинтересоваться, понравилась ли им она. Часто они не могут вспомнить, какие действия совершали, или не хотят рассказывать о возникшей проблеме, стыдясь, что не смогли с ней справиться, или не хотят обидеть программиста, сказав ему, что результат последних двух лет его профессиональной деятельности — полное барахло. (Последней проблемы у меня, как вы догадываетесь, не возникает.)

Чтобы выяснить ситуацию, программист должен знать точные действия пользователя при его взаимодействии с интерфейсом. С чего начинает пользователь? Каковы его последующие действия? Сколько раз ему нужно попытаться что-то выполнить, прежде чем он разберется, как правильно действовать? Сколько времени требуется, чтобы обратить внимание на некую функцию?

При этом наблюдение за пользователем не должно оказывать влияния на его поведение. Это означает, что пользователь должен находиться в отдельном помещении и располагать только теми вспомогательными средствами (например, электронной документацией или Google), которые у него будут в реальных условиях. Наблюдать за пользователем нужно через зеркальную стену, вести видеозапись действий и регистрировать их программным образом, чтобы знать, какие он нажимал клавиши и что делал мышью при работе с приложением. В некоторых лабораториях даже применяют устройства для слежения за направлением взгляда пользователя, смотрящего на экран.

Когда вы это сделаете, на вас снизойдет просветление. Алан Купер в своей классической книге о проектировании интерфейса пользователя «About Face: The Essentials of User Interface Design» (IDG Books, 1995) пишет: «[Профессионалы юзабилити] затаскивают программистов в темную комнату, где они смотрят через зеркальное окно, как несчастный пользователь сражается с их программой. Поначалу программисту кажется, что у пользователя какое-то заболевание мозга. Наконец, в результате длительного и мучительного наблюдения программист вынужден отступить перед фактами. Он признает, что интерфейс пользователя нуждается в доработке и торжественно обещает ее выполнить».

К сожалению, слишком часто тестирование юзабилити откладывается до поздних этапов разработки, едва ли не до самой поставки продукта. Как правило, работа отстает от графика, поэтому тестирование юзабилити часто вообще опускают. Если же тестирование дает полезную информа-

цию, то в графике работ часто уже не остается времени на исправления.[1] Тестирование юзабилити следует проводить рано, в идеале вообще до начала программирования.

В некоторых компаниях полагают, что широкие испытания перед самым выпуском позволяют получить продукт с лучшими потребительскими свойствами. Например, в Microsoft принято «съесть то, чем собираешься кормить свою собаку» («dog-fooding»). Для этого перед выпуском продукта в свет его передают реальным пользователям внутри компании, например пересаживают секретарш на новую версию Word для Windows. При этом выявляются некоторые ошибки, но они оказываются логическими ошибками программирования, типа арифметических, нарушающими работу программы. Искать конструктивные ошибки, особенно в сфере юзабилити, слишком поздно. Да, собачий корм будет немного вкуснее, чем без такой проверки. Но он не превратится в корм для кошек, а проверка происходит слишком поздно, и проверяющие не успевают выяснить, что потребителями корма станут не собаки, а кошки, а то и жирафы.

А вот пример правильного подхода. Однажды я консультировал страховую компанию, в которой писали Windows-программу, чтобы заменить дорогие терминалы IBM. Довольно необычно для страховой компании, что они провели испытание юзабилити, о котором я сейчас рассказывал. И они все сделали грамотно — с видеозаписью и наблюдением через зеркало. Выяснилось, что пользователи в целом довольны приложением и удовлетворены его работой. Но они по привычке нажимали Enter для перехода из одного поля ввода в другое, как это было принято в терминалах IBM, вместо клавиши Tab, как в приложениях Windows. Разработчикам предложили изменить поведение программы. После тщательного размышления разработчики решили, что хотя технически такое изменение сделать очень легко, оно не сильно облегчило бы жизнь пользователям, вопреки надеждам последних. Конечно, можно было заставить приложение работать по-старому. Но все новые коммерческие приложения Windows, которые планировалось устанавливать, будут работать по-другому, и пользователи сойдут с ума, переключаясь с одной схемы на другую по

[1] В одной из своих книжек для программистов я сформулировал закон Платта «экспоненциального роста оценки», который гласит, что «всякий программный проект требует втрое больше времени против самой тщательной оценки, даже если она получена с учетом этого закона».

нескольку раз на дню. Разработчики все же убедили пользователей смириться с изменением привычного стиля. После неизбежного периода протестов пользователи успокоились, чему способствовало и ужасное положение на рынке труда в тот момент. Я не утверждаю, что программисты должны впихивать в глотку пользователям то, что тем должно быть полезно. Обычно это не удается, но здесь был особый случай. Я привожу эту историю с моим клиентом как пример хорошего тестирования юзабилити. Они провели необходимое тестирование. Они получили результаты. И они сделали правильные выводы из полученных результатов. Почаще бы компании так поступали.

Какова ситуация и что можно сделать

Что из этого следует для нас, бедных пользователей? Подытожу сказанное:

1. Вы *не* глупы. Интерфейсы пользователя действительно скверные, чего быть не должно.
2. Плохи интерфейсы, потому что их разрабатывают программисты, не понимающие разницы между собой и пользователями.
3. По причине, указанной в п.2, интерфейсы намеренно усложнены, и авторы считают, что пользователям это должно нравиться, что является ошибкой (см. п. 1).

Интерфейс пользователя можно сделать гораздо лучше, если привлекать специалистов по юзабилити в самом начале каждого программного проекта. Обычные программисты в этом отношении малопригодны, чтобы не сказать хуже. Кто-то должен встать на защиту молчаливого большинства пользователей, которым глубоко наплевать на технологию как таковую, которым просто нужно выполнить задачу и после этого заниматься своими обычными делами. Это я и пытаюсь делать на всех обсуждениях дизайна, где мне приходится бывать. «Вы, как те конструкторы дрелей для сети Home Depot, — говорю я программистам, — спорите, что лучше: шариковые подшипники, роликовые или пневматические, и каждый утверждает, что именно его решение нужнее всего на свете покупателю. Все не так. Покупателю не нужна ваша дрель сама по себе, нисколечко. Не была нужна и никогда не будет. Он идет в Home Depot не потому, что ему нужна дрель, а потому что ему нужны отверстия. Если бы он мог купить коробку отверстий для своей стены, он бы и не касался вашей дрели

и был бы вполне счастлив. (Помните Ринго в *Yellow Submarine*? «I've got a hole in my pocket...» — «у меня в кармане дырка».) Ваша дрель — лишь неизбежное зло на пути вашего пользователя к отверстиям. А вот теперь спросите себя сами и честно ответьте: какие отверстия нужны вашему пользователю и как ваша программа может помочь ему получить их лучше, быстрее и дешевле?»

Прочтя эту главу, вы можете не хуже всякого другого объяснить производителю программного обеспечения, что вам нравится, а что нет. Структура интерфейса программы — это не откровение свыше, дарованное на горе Синай. Она создается на основе конструктивных решений программистов и других разработчиков, и эти решения вполне могут быть иными. Пишите им электронные письма, и почаще. Объясняйте, что вам нравится и что нет. Требуйте, чтобы они не совали вам под нос этот запрос подтверждения, а реализовали бы хорошую функцию Undo.

Больше всего на свете программисты боятся выглядеть глупыми. Они готовы быть уродами, импотентами, мучителями детей и животных, но только не тупыми. Да смилостивится к нам Господь. В их представлении о себе интеллект ценится превыше всего. Если вы хотите, чтобы программист что-то сделал, постарайтесь, чтобы он глупо выглядел на публике, если не сделает этого.

Итак, когда в очередной раз вы встретите плохой интерфейс пользователя, остановитесь и рассмотрите его. Поработайте с ним немного, выясните точно и конкретно, чем он вам не нравится и чего бы вам хотелось от него. Пошлите сообщение о плохом дизайне в Зал позора «Hall of Shame» — веб-сайт, специально созданный для таких целей. Можно начать с веб-сайта этой книги *www.whysoftwaresucks.com*. Затем пошлите письмо компании-изготовителю с извещением о том, как вы их выставили на позор. Чем крупнее их глупость, обнаруженная вами, тем больше они будут раздосадованы ее публикацией. Тогда, *возможно*, до них дойдет, что их пользователи не такие, как они сами.