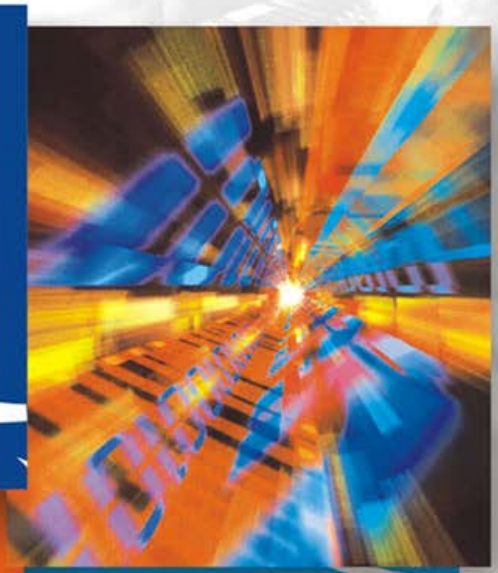
ЮРИЙ РЕВИЧ

bhv

НЕСТАНДАРТНЫЕ ПРИЕМЫ ПРОГРАММИРОВАНИЯ НА DELPHI



- РЕЗИДЕНТНЫЕ ПРОГРАММЫ В WINDOWS
- ИНСТАЛЛЯЦИЯ И ДЕИНСТАЛЛЯЦИЯ ПРОГРАММ
- ПОИСК В ДОКУМЕНТАХ
- РАБОТА С ГРАФИКОЙ В WINDOWS
- ЛЮБИТЕЛЬСКАЯ КРИПТОГРАФИЯ
- РАБОТА С COM- И USB-ПОРТАМИ

PRO

ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

УДК 681.3.06
ББК 32.973.26-018.2
Р32

Ревич Ю. В.

Р32 Нестандартные приемы программирования на Delphi. — СПб.: БХВ-Петербург, 2005. — 560 с.: ил.

ISBN 5-94157-686-2

Книга призвана помочь программистам разрабатывать полноценные, профессиональные Windows-приложения в Delphi. Показано, как предотвращать повторный запуск приложения, работать с нестандартными окнами, перехватывать нажатие клавиш, создавать резидентные программы в Windows, а также инсталляторы и деинсталляторы программ, осуществлять поиск в документах, работать с COM- и USB-портами, шифровать текст и многое другое. Рассмотрены примеры решения этих и многих других проблем, которые встают при создании программы, ориентированной на долговременное использование и распространение. Приведены приемы работы с Windows API. Изложение ведется на примерах поэтапного создания реально работающих практических приложений. Компакт-диск содержит исходные тексты разобранных в книге примеров.

Для программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 21.09.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 45,16.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-686-2

Оглавление

ГЛАВА 1

Ликбез

Некоторые сведения о программировании, Windows и Delphi

> Создайте систему, которой сможет пользоваться каждый дурак, и только дурак захочет ею пользоваться.
>
> *Принцип Шоу*

> Компьютерная программа выполняет то, что вы приказали ей делать, а не то, что вы бы хотели, чтобы она делала.
>
> *Третий закон Грида*

Современный персональный компьютер — устройство необычайно сложное. Причем рядовой ПК образца 2004 года отличается от IBM PC двадцатилетней давности гораздо больше, чем, к примеру, последняя модификация Ford Focus от легендарного Ford T образца 1913 года. И дело тут не в самой по себе скорости работы, которая возросла примерно на три порядка. Гораздо важнее принципиально возросшая функциональность — ни о какой 3D- и даже обычной многоцветной 2D-графике тогда и речи не шло, голосовые интерфейсы существовали разве что в фантастических романах, а предсказать нечто подобное Интернету, да еще и мобильному, фактически не смог никто. Если продолжить аналогию с первыми автомобилями, то современный ПК в сравнении с IBM PC скорее следует уподобить, если и не реактивному истребителю, то, по крайней мере, пассажирскому лайнеру.

Казалось бы, управлять такой сложной машиной — учиться и учиться. Но ПК не обосновались бы настолько прочно в наших домах и офисах, если бы разработчики не придумали программные средства, сводящие сверхсложные операции к двум-трем щелчкам мыши. Причем на сегодняшний момент сложилась довольно парадоксальная ситуация: разнообразие компьютерного "железа" настолько велико, что *правильно* подобрать комплектующие и *правильно* настроить современный ПК невозможно без определенного багажа специальных знаний. А вот технологии программирования ПК, наоборот, предельно упростились. Правда, это в полной мере справедливо только для

случая создания пользовательских программ — системные программы, разумеется, требуют для своего создания специальных знаний. Показательно, что в популярной книге С. В. Зубкова "Assembler для DOS, Windows и Unix" [13], выдержавшей несколько переизданий, программированию под Windows и Unix посвящено менее 100 страниц из почти 700 — настолько это простое по сути занятие.

Заметки на полях

Попробую пояснить эту парадоксальную мысль. Для начала хочу подчеркнуть разницу между понятиями "сложный" и "громоздкий" — очень простая по сути программа может быть очень громоздкой и наоборот. Например, расшифровать заголовок файла, содержащего изображение в формате BMP, и воспроизвести на экране содержащуюся в нем картинку — достаточно сложное занятие. Программа, которая это делает из-под DOS, окажется и сложной и достаточно громоздкой, учитывая еще тот факт, что для воспроизведения True Color придется где-то искать и присоединять к программе драйвер имеющейся в наличии видеокарты. Излишне говорить, что в силу большого разнообразия "железа" последняя задача может доставить и программисту и пользователю массу хлопот. А вот в Windows ни один из этих вопросов не стоит вообще — все драйверы уже установлены заранее, а для вывода на экран картинки формата BMP не требуется даже знать, что у нее есть какой-то там заголовок — даже на ассемблере это делается вызовом одной-двух функций. Вместе с тем, если простейшая ассемблерная программа под DOS может состоять из одной-единственной команды процессора, то любая программа под Windows обязана содержать некий минимум команд, так что решение упомянутой ранее задачи отображения картинки по объему кода может даже превысить DOS-вариант. Что делает Windows-программы более громоздкими, но отнюдь не более сложными, потому что этот необходимый минимум повторяется из программы в программу и заново его "изобретать" каждый раз не требуется (см. [13]).

Так получилось потому, что большинство функций в современных операционных системах скрыто за оболочкой, носящей название *пользовательские программные интерфейсы* (Application Programming Interface, API). Собственно процесс программирования сводится к тому, чтобы вовремя и правильно вызвать нужную функцию API. Поэтому такие крайне сложные для "обычного" программирования операции, как, к примеру, вывод на экран полноцветной графики или создание текстового редактора с поддержкой форматирования, шрифтов, операций с буфером обмена и прочих необходимых свойств, сводится к вызову нескольких функций API. Довольно большой прогресс по сравнению с прерываниями DOS, которые программисты в большинстве случаев старались обойти, не видя необходимости в лишних прослойках между программой и BIOS или "железом", не так ли?

Однако на этом разработчики современных систем не остановились. С помощью средств так называемого *визуального программирования* процесс вызова функций API максимально автоматизирован. Вам уже не нужно выполнять рутинную работу по написанию кода программы целиком. Достаточно

перетащить мышью нужный компонент на форму, и соответствующий код включается в текст программы автоматически. В результате процесс программирования стал воистину творческим занятием: всю "грязную" работу берет на себя компьютер (точнее, выбранный пакет программирования), а вам остается только правильно обработать возникающие в программе события. Специальных знаний об устройстве компьютера и построении ОС здесь требуется не больше, чем для обычного квалифицированного пользователя.

Разумеется, как в любом другом деле, за удобства приходится платить: вы полностью привязаны к существующим программным интерфейсам, которые писали для вас добрые дяди из Microsoft и других фирм-производителей системного софта, и шаг вправо-влево тут означает если и не расстрел, то, во всяком случае, необходимость приобретения *очень специальных* знаний. Однако такой выход за пределы стандартных функций в подавляющем большинстве случаев и не требуется: "добрые дяди" постарались предусмотреть по максимуму все, что требуется среднестатистическому пользователю. Другое дело, что поиск нужной функции и способа ее правильного использования может быть очень непростым занятием — ведь далеко не все можно сделать, перетаскивая компоненты мышью на форму. Именно в этом деле и призвана помочь книга, которую вы держите в руках.

О Delphi и Windows

Кстати, а почему именно Delphi? В принципе все современные пакеты визуального программирования позволяют делать одно и то же, а если какие-то вещи делать удобнее в одном пакете, а другие — в другом, то никто не мешает использовать их совместно. Скажем, библиотеки VCL (Visual Components Library) от Borland являются общими для Visual C++ и Delphi и написаны в основном на Object Pascal, а Windows API, наоборот, большей частью написаны на C (или C++), что не мешает использовать их в любой среде. Но Delphi, безусловно, является на сегодняшний день наиболее универсальной средой программирования, которая позволяет без лишних сложностей создавать как самые простые пользовательские программы, так и навороченные профессиональные пакеты.

Эта книга написана с расчетом на выполнение примеров в среде Delphi 7.0. Седьмая версия — последний релиз Delphi для платформы Win32, на которой основываются все версии Windows от 95-й до XP. В настоящее время Microsoft переходит на платформу .NET — на ней будет полностью основана Windows Longhorn, которую планируется выпустить на рынок в 2006 году. Частично на .NET переходит уже обновленная 64-разрядная Windows XP, готовящаяся к выходу в 2005 году. Между прочим, базовый язык программирования для платформы .NET под названием C# разрабатывает Андерс

Хейлсберг — человек, которому корпорация Borland во многом обязана самим своим существованием.

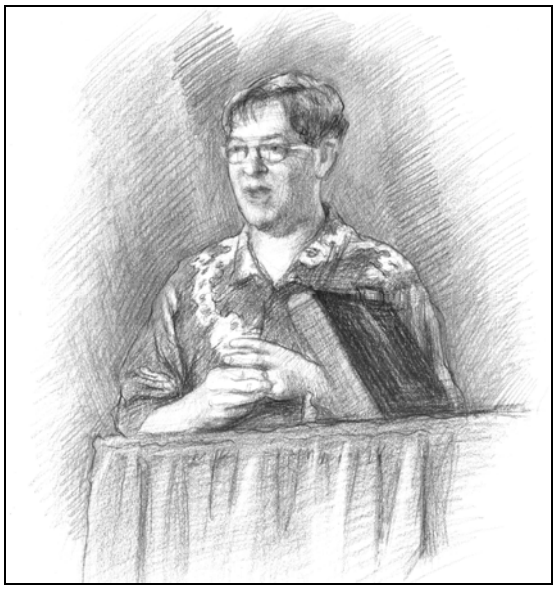


Рис. 1.1. Андерс Хейлсберг (рисунок Александры Дрофиной)

Андерс Хейлсберг (Anders Heilsberg) — создатель Turbo Pascal и один из главных архитекторов Borland Delphi с момента ее возникновения, человек, разработавший продукты, выведшие корпорацию Borland в ряды ведущих поставщиков программного обеспечения. Также главный архитектор языка C# для платформы Microsoft .NET. О себе Андерс рассказывает так (из интервью журналу "Домашний компьютер", #1, 2004):

Я родился в 1960 году в Копенгагене, Дания. Свое тихое и милое детство я провел в пригороде Копенгагена. Потом учился на инженера по электротехнике в Техническом университете Дании. В 1987 году переехал в США, а в 1994 — женился. Живу в Сиэтле. С 1983 по 1996 я работал в компании Borland, а теперь в Microsoft. В 1979 я основал компьютерную компанию в Дании под названием PolyData. Это было время, когда персональных компьютеров еще не существовало. Мы продавали компьютерные комплексы и писали для них программное обеспечение. Я написал такие вещи, как ассемблер, дизассемблер, небольшую операционную систему и несколько расширений для Microsoft ROM-Basic. Моим самым первым большим проектом стал компилятор с языка Pascal и редактор, который мог заменить ROM-Basic. После этого я написал еще одну реализацию Pascal для операционной системы CP/M. Она называлась PolyPascal. В 1983 году мы объединились с

> *ребятами, которые только что основали компанию Borland, они лицензировали наш компилятор Pascal, добавили туда свой собственный редактор и назвали все это Turbo Pascal. Я помню, как думал, что они сумасшедшие: эти парни продавали новый продукт по цене 49 долларов 95 центов, в то время как он стоил 500 долларов! Но достаточно быстро выяснилось, что я ошибался — Turbo Pascal стал очень популярным. Мы продали его столько, что в начале было невозможно представить.*

Для того чтобы читатель немного сориентировался, приведем краткую сравнительную характеристику различных сред обработки от Borland, базирующихся на языке Pascal.

В начале всего был Turbo Pascal 1.0, вышедший на рынок 20 ноября 1982 г. Об искусстве Андерса Хейлсберга может говорить тот факт, что интегрированная среда разработки, встроенный редактор и библиотека времени выполнения умещались в файле turbo.com размером 33 280 байт. Правда, эта версия сама могла делать только COM-программы (если кто помнит, то в DOS существовал такой формат исполняемых файлов, отличавшийся от привычного EXE тем, что занимал только один 64-килобайтный сегмент памяти), но зато работала она на медленных ПК того времени очень быстро и давала очень компактный код, благодаря чему сразу вывела компанию Borland в лидеры. В дальнейшем эволюция Turbo Pascal привела к версии 7.0 (1992) — до сих пор популярному средству разработки программ под DOS. Благодаря непревзойденной по удобству среде разработчика (Integrated Development Environment, IDE) и простому для освоения языку, Turbo Pascal стал особенно популярным в непрофессиональной и полупрофессиональной среде. Строго говоря, последняя версия Turbo Pascal, как такового, называлась 6.0, седьмая версия называлась Borland Pascal и включала в себя две разных среды: Borland Pascal for DOS 7.0 (базирующийся на шестой версии с некоторыми существенными доработками) и Borland Pascal for Windows. Если вам удастся достать какое-нибудь пособие по последнему (например, раритетную книгу В. В. Фаронова "Паскаль и Windows"), то вы увидите, что разработка приложений под Windows "вручную" — довольно громоздкое занятие, требующее хорошего знания приемов объектно-ориентированного программирования (ООП). Чтобы облегчить это занятие, в Borland (под руководством все того же Хейлсберга) к 1995 году была разработана Delphi 1.0 — первая визуальная среда программирования, ориентированная на разработку 16-разрядных приложений под Windows 3*x*.

Начиная же с версии 2.0, Delphi ориентировалась на 32-разрядные версии Windows (95 и выше), использующие платформу Win32. Все использующиеся на практике на момент выхода этой книги версии Windows ориентируются

именно на эту платформу, поэтому в подавляющем большинстве случаев вам вообще не нужно думать о том, какая версия Windows у вас установлена. И все же многие API могут быть специфичными для того или иного семейства ОС либо той или иной версии. Поэтому работу программ, особенно тех, которые предназначены для распространения, следует проверять под различными версиями. Для этого удобно установить на компьютере сразу несколько систем, благо XP это позволяет без лишних сложностей.

На момент написания данной книги уже имеются версии Delphi 8.0 и Delphi 2005, ориентированные на .NET. Для совместимости в комплект поставки Delphi 8.0 входит Delphi 7.1, подобно тому, как в Borland Pascal входил Turbo Pascal, а Delphi 2 комплектовалась версией 1. Разница только в том, что вторая версия Delphi в свое время вышла с опозданием, уже после выхода Windows 95, а восьмая версия отчасти опережает события. Заметим, что наименование "Delphi 8.0" следует признать неудачным — это другой продукт для другой платформы (изменили же когда-то марку Turbo Pascal на Borland Pascal — почему бы и не следовать этому принципу и в дальнейшем?). Поэтому большинство примеров, которые вы встретите в этой книге, в среде Delphi 8.0 (и выше), скорее всего, просто не будут компилироваться — но сами по себе готовые программы будут пригодны еще долгие и долгие годы, в будущей Longhorn декларирована совместимость с Win32 (подобно тому, как версии Win95/98/ME были совместимы с DOS-программами).

Что же касается более ранних версий, то большинство примеров из этой книги совместимы с версиями Delphi, начиная с 4.0, в крайнем случае 6.0. При установке следует позаботиться, чтобы у вас оказались установленными файлы справки по Win32. Эта справка, к сожалению, практически не обновлялась с версии 3.0, так что можно просто скопировать файл win32.hlp в отдельный каталог и пользоваться им автономно, независимо от версии пакета.

Примечание специально для тех, кто только осваивает программирование: на пиратских развалах якобы появилась русифицированная версия Delphi. Ее устанавливать *не следует*, и не только потому, что она неизвестного происхождения, но и потому, что все без исключения пособия и интернет-ресурсы оперируют с английской версией, и в результате вы потратите гораздо больше времени на обратный перевод русских названий пунктов меню и текстов сообщений, чем на то, чтобы один раз выучить английские.

> Несколько слов о том, каковы особенности работы программ под теми или иными версиями Windows. Главное отличие семейства 9*x* (95/98/ME) от семейства NT (NT/2000/XP) заключается в реализации многозадачности. В забытых ныне 16-разрядных версиях Windows 3.*x* многозадачность называлась кооперативной — фактически весь процесс работы заключается в выполнении последовательного ряда событий Windows, и пока одно событие

не обработается, все остальные вынуждены ждать своей очереди. Таким образом, малейшая ошибка в одном приложении полностью подвешивает всю систему. Все программы, в том числе служебные, видимы друг для друга, модуль, содержащий ошибки обращения к памяти, может легко испортить ее содержимое, принадлежащее другому процессу.

В версиях семейства NT реализована настоящая — вытесняющая — многозадачность (напомним, что NT появилась раньше, чем 95-я). Такая система распределяет процессорное время и память между процессами (process) и потоками (thread)[1], как будто бы каждый из них выполняется в отдельном компьютере, поэтому "зависшая" программа теоретически не оказывает влияния на все остальные. Но за это приходится платить — в первую очередь тем, что прямой доступ к "железу" практически исключен, а если вы попытаетесь этот запрет обойти тем или иным способом, то такая попытка ничем хорошим ни для вашей программы, ни для системы не кончится. То есть требования к качеству программного кода сильно повышаются и можно смело утверждать, что все случаи "обрушения" Windows XP связаны именно с неправильно (как говорят программисты — "криво") написанными приложениями.

А в версиях 9*x* в целях совместимости с 16-разрядными приложениями (включая DOS-программы) реализовано что-то вроде промежуточного варианта. Все 32-разрядные прикладные программы выполняются в соответствии с моделью вытесняющей многозадачности. На уровне таких приложений формально все работает независимо, как и NT. Однако в области модели памяти (в основном из-за требований совместимости с 16-разрядными приложениями DOS и Win3*x*) есть серьезные дыры. Как и в 3*x*, ничто не может помешать программе, содержащей ошибку обращения к памяти, произвести запись в адреса, принадлежащие системным DLL, и вызвать крах всей системы.

В Windows 9*x* ситуация, когда выполняющийся поток все никак не освободит ресурсы компьютера, случается не так уж редко. Даже формально правильное приложение может быть источником неприятностей: так, многим знакома картинка, когда ресурсоемкая программа (архиватор, файловый менеджер при копировании больших объемов информации, или, например, поисковая программа) настолько "оккупирует" все ресурсы компьютера, что даже картинка рабочего окна не успевает прорисовываться. Один из механизмов таких задержек — но не единственный, конечно — связан с тем, что многие подобные процессы выполняются через 16-разрядные функции, а

[1] Сами по себе процессы ничего не делают, а лишь предоставляют ресурсы и контекст для выполнения потоков. Любой процесс должен иметь, по крайней мере, один поток, который и выполняет код процесса. Причем планирование переключения задач в Windows осуществляется на уровне потоков, а не процессов.

только один поток может обращаться к 16-разрядным DLL в каждый момент времени, потенциально затормаживая другие процессы, которым нужен к ним доступ. Другой — с неправильным распределением приоритетов одновременно выполняющихся потоков. Бороться с такими неприятностями можно только внутри самой программы, периодически искусственно вызывая обработчик системных сообщений, но самое правильное — по возможности не использовать потенциально "тормозных" процедур вообще.

О пользовательских интерфейсах компьютерных программ

Крупнейший в мире специалист по компьютерным интерфейсам Джеф Раскин (Jef Raskin, 1943—2005 гг.) говорил примерно так (за точность цитаты не ручаюсь): *"Пользователь обычно не понимает, насколько ему неудобно, механизм обратной связи от потребителя к производителю не работает, и единственное, что может заставить проектировщика создавать по-настоящему хороший интерфейс — это его совесть"*. Я не могу тут не остановиться на истории пользовательских интерфейсов, потому что непонимание, насколько неудобно работать с современными программами, к сожалению, характерно для программистов в еще большей степени, чем для пользователей. Невозможно делать удобные программы, если вы не очень хорошо представляете, с чем вам придется иметь дело.

Точкой роста почти всех идей, реализованных впоследствии в так называемом графическом интерфейсе пользователя (Graphic User Interface, GUI), стал Xerox Palo Alto Research Center (Xerox PARC), возникший на рубеже 60—70 годов. Любопытно, что одним из его основателей, а с сентября 1970 года — и руководителей, стал Боб Тейлор (Robert W. Taylor), который пришел в Xerox из DARPA, где в 1966—69 гг. возглавлял проект ARPAnet, предшественника Интернета. Одной из идей, родившихся в этом центре, была так называемая парадигма WIMP (Windows, Icons, Menus, Point-and-Click — "окна, пиктограммы, меню, укажи и щелкни"), которая переросла позже в концепцию GUI и продолжает эксплуатироваться в настоящее время. Эти разработки связывают с именем Алана Кея (Alan Key), также известного, как автора SmallTalk (первого объектно-ориентированного языка программирования). В 1975 году в Xerox была начата разработка нового проекта, закончившегося в апреле 1981 года представлением системы Xerox 8010 Professional Workstation, более известной под торговой маркой Xerox Star. Именно с нее и началось победное шествие GUI, внедрение которого, как промышленного стандарта, было произведено в 1988 году усилиями Sun Microsystems, Xerox и AT&T. Xerox Star мы также обязаны такими вещами, как иконки, окна, меню, двухбайтовые многоязычные шрифты (современный Unicode — см. *главу 8*), режим

WYSIWYG и др. За подробностями я отсылаю читателя к [25], а здесь нам важно, что распущенный после рыночного провала Xerox Star примерно в 1983 году коллектив ее разработчиков в основном оказался в Apple — за важными исключениями, о которых поговорим отдельно.

В числе прочих инноваций в интерфейсе Star было впервые использовано представление экранного пространства в виде "рабочего стола" (Desktop). Эта метафора основывается на том, что пользователь якобы не имеет представления о существовании, скажем, программы под названием "текстовый редактор", а просто "открывает документ". Альтернативой является концепция "инструментов" (Tools) или "приложений" (Applications), где пользователь запускает нужный инструмент (приложение), и с его помощью открывает документ, причем тип его идентифицируется обычно по расширению имени файла. Ответственность за результат, например, загрузки файла, содержащего изображение, в текстовый редактор при этом целиком ложится на пользователя. Разработчики Star сознательно шли на потерю универсальности, присущую инструментальной модели, в то же время предупреждая об ограниченности сферы применимости метафоры "рабочего стола" исключительно офисными системами. (Дуглас Энгельбарт, изобретатель мышинного интерфейса, позднее иронизировал: *"Весь мир был увлечен идеей "офисной автоматизации", будучи уверен, что "настоящие пользователи" компьютеров — это секретари, чьи задачи необходимо автоматизировать."*)

В 1978 году упомянутый ранее Джеф Раскин работал в Apple, где начал новый проект под названием Macintosh, однако уже в 1982 г. разругался с Джобсом и покинул фирму. Впрочем, это неудивительно — ни один из его проектов так и не стал успешным в коммерческом смысле, притом что выдвинутые им теории обязательно изучают на всех соответствующих курсах в мире. Основная заслуга Раскина в том, что он одним из первых осознал: самое важное ментальное ограничение человека — ограниченность внимания. Фокус внимания у человека один. Интерфейсы, переключающие на себя внимание человека, — это одна из ключевых причин неэффективности взаимодействия с машинами.

Заметки на полях

Вот пример: когда мне диктуют телефонный номер, я хочу сразу записать его, пока он не вылетел из памяти. В типичной современной ОС для этого требуется запустить специальное приложение (по крайней мере, переключиться на него), после чего создать новый контакт (это же надо еще знать, что то, что вам нужно, называется именно "контактами", см. *главу 16*), найти на появившейся форме подходящее поле, вписать туда номер... Автор этих строк и в бумажной-то телефонной книжке не вел записи по алфавиту, предпочитая для ускорения записывать их "внавал", и сейчас фиксирует номера, адреса и прочие полезные сведения в маленькой программке в стиле картотеки из Windows 3*x*. И даже забросил ведение адресной книги в The Bat! — слишком много усилий приходится

предпринимать для систематизации записей и извлечения оттуда нужного адреса, проще найти корреспондента через "Поиск".

Раскин выступил с радикальным предложением: отказаться от зоопарка разных приложений и заменить его на единую рабочую среду, которая всегда ведет себя одинаково. В такой среде, например, нажатие клавиш всегда приводило бы к вводу текста, поэтому человеку, желающему записать телефон, достаточно было бы просто его набрать. Разумеется, определить введенный номер в "правильное" место нужно было бы и в этой среде, но в ней это можно сделать после ввода номера, что принципиально удобнее. В результате Раскин придумал интерфейс, суть которого состоит в следующем: все, что хранится в компьютере, представляет собой единый документ, отдельные приложения — это команды и модули, запускаемые пользователем или подключающиеся автоматически. Причем все управление осуществляется с помощью клавиш — мыши Раскин не признавал, действительно, ведь для работы с текстом мышь в принципе не требуется, так зачем лишние сущности? Несмотря на понятные ограничения (а кроме инструментальной модели, предоставляющей пользователю полный доступ к "потрохам" системы, вплоть до ее перепрограммирования, любая другая модель является ограниченной и пригодной лишь в определенной области) эта система для тех, кто работает именно с документами, была бы, вероятно, более удобной. И в конце 80-х она даже была осуществлена на практике в компьютере Canon Cat, который разошелся в количестве 20 тыс. экземпляров, но затем проект был свернут.

Подход Раскина можно интерпретировать так: вся среда в компьютере есть одно универсальное приложение. Легко заметить, что черты этого подхода можно встретить, например, в MS Office, в попытках интегрировать все действия пользователя — или большинство их — в единую среду. Я не знаю, что могла бы представлять доведенная до логического завершения концепция Раскина (в Canon Cat был осуществлен фактически лишь текстовый редактор с функциями электронных таблиц), но есть сильное подозрение (основанное на практическом опыте), что ни один программист, даже такой корифей, как Джеф Раскин, не сумел бы сделать интегрированную среду во всех нюансах так, чтобы в ней было бы удобно работать всем без исключения.

Но вернемся к существующим интерфейсам. Для платформы PC (уже превратившейся к тому времени в Wintel) "рабочий стол" был реализован только спустя пятнадцать лет — с появлением Windows 95. Казалось бы, было время все продумать и учесть недоработки Xerox и Apple, но в Windows была предпринята эклектичная попытка сохранить все преимущества инструментальной модели, как наиболее гибкой, но рассчитанной на специально обученных пользователей, и в то же время склонить на свою сторону армию "чайников", обучаться не желающих или не имеющих возможности. В результате пророческие слова Раскина о том, что *"пользователь обычно не понимает, на-*

сколько ему неудобно" относятся к Windows, как ни к чему другому. Буквально любое незнакомое действие в Windows, несмотря на все попытки унификации и стандартизации, требует от "чайника" консультаций с более продвинутым собратом по несчастью[2]. Непоследовательное использование Desktop-метафоры может приводить к просто катастрофическим последствиям: т. к. первичным признаком для отнесения файла к тому или иному типу в Windows по-прежнему служит расширение его имени, то при неправильном переименовании файл может быть просто потерян для непросвещенного пользователя. При попытке скрыть расширения, как это происходит по умолчанию в Windows, часто может возникать забавная ситуация, когда в одной папке оказываются несколько абсолютно одинаковых с виду значков, что приводит к недоразумениям — одно дело послать по электронной почте многомегабайтный TIFF, другое — компактный JPEG, а ведь все иконки, относящиеся к изображениям (если вы не зададите специально обратного), будут связаны с одним приложением, и потому и обозначатся одинаково.

Рискну предположить, что правильным решением всех этих проблем был бы выпуск множества относительно автономных модификаций одной и той же ОС с интерфейсами, "заточенными" под нужды бухгалтера, секретаря, технического писателя, ученого, фотохудожника, журналиста, ребенка, наконец, с возможностью установки их в любой комбинации и переключения между ними. Глядишь, и модель Раскина вполне тогда вписалась бы в коммерческие продукты, и любителям прыгающих по экрану кнопок или "скрепок-помощников" тоже было бы где развернуться...

> Резюмируем то, что сказано ранее, попробовав сформулировать основную цель любого интерфейса — не только компьютерного. Подавляющее большинство действий совершается человеком бессознательно. Часть подобных действий человек умеет совершать от рождения, таких, как глотательные рефлексы или отдергивание руки от горячего предмета. Среди этих действий есть очень сложные *программы поведения* — например, половое поведение или стайные инстинкты. Другая часть — инстинкты и программы, приобретенные в процессе взросления и обучения, это навыки ходьбы, речи, письма и счета, или социальное поведение: стыдливость, способность к состраданию и т. п. Наконец, сравнительно небольшую часть действий составляют те, что контролируются сознанием. Однако именно эти действия требуют повышенного внимания, они человека, пользуясь расхожим выражением, "поглощают целиком". Ни один человек не может обдумывать две

[2] Причем по мере развития одни неудобства устраняются (так, реализация Plug&Play в Windows XP принципиально лучше всех предыдущих версий), зато множатся другие (в той же XP — непонятные проблемы с языком, атрибутами папок, активацией и т. п.).

вещи сразу. Поэтому любое обучение всегда преследует одну и ту же цель — перевести некие действия в область бессознательного, сделать так, чтобы они выполнялись автоматически. Именно на это направлены бесконечные тренировки спортсменов, балерин, специалистов по рукопашному бою, водителей автомобилей. Более того, на это также направлены и процессы умственного обучения — студентов заставляют решать учебные задачи с тем, чтобы в процессе практической деятельности нужное решение находилось как бы "само", на основе приобретенного опыта.

Интерфейс GUI менее всего приспособлен к эффективному обучению. Производители затвердили словосочетание "интуитивно понятный" применительно к интерфейсам, как будто это вообще что-то означает — интуитивно понятным является только тот интерфейс, к которому вы привыкли, и он не заставляет вас задумываться над каждым телодвижением. Обратите внимание, как ловко подростки манипулируют предельно "неинтуитивным" интерфейсом мобильников, и вы поймете, что это совершенно пустое понятие. Интерфейс может и должен быть *логичным* — хотя и это, в общем, имеет значение только для удобства обучения, но уж к "интуитивности" точно не имеет никакого отношения. Является ли "интуитивно понятным" интерфейс управления автомобилем? Отнюдь нет — обратите внимание, скажем, что сцепление нужно *нажимать*, чтобы его *выключить*. Или: если вы замрете при ходьбе, то остановитесь, а если вы все бросите в процессе езды, то автомобиль вовсе не остановится, как можно было бы предположить, исходя из *интуитивного* представления. Но никто ведь не протестует, правда? Потому что в принципе неважно, какие именно действия нужно вызубрить, "заложить в подкорку" (на самом деле — скорее в мозжечок) — важно, чтобы они не требовали в дальнейшем напряжения сознания. Если вы будете все время в процессе езды раздумывать над вопросом, какую педаль нужно нажать для снижения скорости — вряд ли вы вернетесь из поездки живыми.

А вот для существующего WIMP-интерфейса вызубрить действия даже в одной отдельно взятой программе нельзя. Кажущаяся простота действий при указании мышью пункта меню на самом деле оборачивается тем, что это действие невозможно автоматизировать — сам процесс таков, что требует полного переключения внимания. Нужно найти это меню зрительно, направить на него мышь, щелкнуть, потом найти нужный подпункт — все это настоящая полноценная задача, требующая обдумывания, занимающая сознание целиком, как требует обдумывания вопрос, куда именно направить автомобиль при движении. Но направление движения — это и есть настоящая цель управления автомобилем, а выбор пункта меню — вовсе нет. Отличный пример на эту тему приводит известный программист Михаил Донской: *"Такой интерфейсный элемент, как линейка прокрутки, находится в противоречии с одним из основных принципов психологии восприятия: у человека может быть только одна точка активного внимания. При использовании же линейки прокрутки приходится смотреть в две совершенно*

различные точки — на прокручиваемое изображение (не пора ли остановиться?) и на линейку. Всем знакомые неприятности с непопаданием мышью в нужную точку при прокрутке или с соскакиванием курсора мыши с линейки — очевидное следствие вышеуказанного противоречия."

Наглядность Windows, которая так привлекает начинающих, в сравнении с интерфейсом командной строки, оборачивается тем, что они и в дальнейшем вынуждены тратить огромное количество времени на обдумывание действий, которые в принципе никакого обдумывания не требуют. Это все равно, как если бы педаль тормоза в автомобиле каждый раз оказывалась на новом месте и ее нужно было бы искать, например, по тому, что на ней имеется табличка: "тормоз". Задача программиста — сделать таким образом, чтобы в идеале *каждое* действие можно было бы выполнять автоматически, "спинным мозгом", сосредоточившись именно на целевой задаче, а не над тем, в каком углу экрана в данный момент находится меню, и каким щелчком мыши — двойным или одинарным — оно в этой программе вызывается. Означает ли это, что надо вернуться к интерфейсу командной строки? Вовсе нет, нужно только четко себе представлять, что запомнить раз и навсегда последовательность нажатия двух-трех клавиш в конечном итоге намного проще, чем каждый раз целиться в меню. А использовать при этом WIMP, инструментально-командную модель, метафоры "рабочего стола", "вселенной" или что-то другое — это уже дело, в принципе, десятое. Главное не забывать, для чего все это делается вообще.

Страна советов

Конечно, в реальности программисту приходится идти на компромисс, подстраиваясь под существующие стандарты. Точно так же, как архитектор или скульптор при своей работе не может не принимать во внимание законы сопромата, программист должен соблюдать некие правила, которые в совокупности делают его творение программным продуктом. Это, разумеется, в первую очередь относится к ПО, которое предназначено для распространения — неважно, за деньги или просто так. Но даже если вы делаете некую программу для собственных нужд, и это не просто единовременная "проба пера", а некий продукт, которым вы собираетесь пользоваться в течение долгого времени, эти стандарты нужно соблюдать. Грубую ошибку делает тот, кто решает за пользователя (даже если этот пользователь — вы сами), в какой последовательности ему нужно нажимать экранные кнопки. Рассчитывать нужно на тот случай, что пользователь справку не читает (даже если она есть) и обязательно первым делом нажмет именно ту комбинацию клавиш, которая "повесит" вашу программу. И вы сами через пару месяцев гарантированно забудете, что и в какой последовательности надо нажимать. Есть несколько общих правил