Налоги в графическом пользовательском интерфейсе

Одна из главных претензий, предъявляемых к графическим интерфейсам опытными пользователями – особенно теми, кто привык к системам с командной строкой, – заключается в том, что осмысленное манипулирование окнами и меню требует дополнительных усилий. Командная строка позволяет просто набрать команду, которую компьютер не медленно выполнит. В оконных системах пользователям приходится открывать различные папки в поисках файла или программы, прежде чем они смогут открыть файл или запустить программу на исполнение.

После появления окна программы на экране приходится растягивать его до нужного размера и перетаскивать в подходящее место.

Все эти претензии имеют под собой основания. Подобные манипуляции с окнами фактически являются налогом. Они не приближают пользователя к намеченной цели, а являются накладными расходами – требованием, выдвигаемым программой прежде, чем она снизойдет до оказания помощи пользователю. Однако каждый знает, что графические интерфейсы проще в использовании, чем системы с командной строкой.

Противоречие возникает из-за того, что реальные механизмы взаимодействия остаются скрытыми. В интерфейсе с командной строкой налоговое бремя пользователей еще тяжелее: прежде чем начать работать с системой, требуется выучить команды. Кроме того, пользователь не может изменить экран по своему вкусу. Налоги командной строки снижаются только после того, как пользователь потратит много времени и сил на ее освоение.

Зато для неопытного, случайного пользователя визуальная ясность графического интерфейса оказывается подспорьем в навигации и изучении возможностей. Пошаговый способ общения пользователя с графическим интерфейсом помогает тем, кто еще не знаком с функциональностью системы. Он помогает в работе и тем, кто должен решать не сколько задач одновременно и запускать сразу несколько приложений.

Налоги и пользователи

Любой пользователь, желающий разобраться в интерфейсе командной строки, может быть автоматически отнесен к категории экспертов. А любой пользователь, разобравшийся в одном интерфейсе командной строки, быстро разберется в любом другом интерфейсе, включая графический. Эти пользователи без труда постигнут любые тонкости обращения с программой. Они запускают программу, имея четкое представление о том, что и как требуется сделать. Такому пользователю только мешает помощь, которую интерфейс предлагает новичку.

Снижая бремя налогов, мы должны быть очень внимательны. Не следует снижать его лишь в угоду опытным пользователям. С другой стороны, нельзя перекладывать все налоги на плечи опытных пользователей, создавая удобства для новичков.

Поддержка новичков и пользователей, работающих с продуктом время от времени, – это та область, где проектировщикам приложений сложно избежать создания излишнего налогового бремени. Легко найти оправдание той функциональности, которая облегчает жизнь новичкам, изучающим программу. К сожалению, эта функциональность быстро превращается в дополнительную нагрузку, когда пользователь перестает быть новичком. Функциональность, добавленная к программе ради обучения пользователей (скажем, пошаговые сценарии), должна легко отключаться. Дополнительные колеса для велосипеда помогают учиться начинающим, но мешают тому, кто уже научился кататься.

Не приваривайте дополнительные колеса к велосипеду намертво.

Типы налогов

«Наглые» налоги

Существуют **действия, в которых не нуждается никто – ни новички, ни специалисты.** Это и есть «наглые» налоги. Операции, связанные с настройкой аппаратной части, например указание программе, какой COM-порт она должна использовать, компьютер мог бы выполнить и самостоятельно. Подобные аспекты следует убирать из пользовательского интерфейса и заменять интеллектуальным поведением программы, скрытым от пользователя.

Визуальные налоги

Визуальный налог – это **работа по расшифровке видимой информации, которую приходится выполнять пользователю**. Поиск конкретного элемента в списке, выяснение того, откуда начинать чтение текста на экране или какие элементы являются активными, а какие служат просто украшениями, – все это визуальные налоги.

Пример значимого источника визуальных налогов – расточительная стилизация графики и элементов интерфейса. Визуальный стиль в первую очередь должен поддерживать прозрачную передачу информации и демонстрацию состояния интерфейса.

В некоторых приложениях украшения могут быть уместны для создания определенного настроения, атмосферы или придания продукту индивидуальности. Однако излишнее украшательство может снизить эффективность работы пользователей, принуждая их расшифровывать различные визуальные элементы, чтобы понять, что является элементами управления и важной информацией, а что служит просто украшениями.

Прекращение работы

Существует особая разновидность налогов, столь распространенная, что заслуживает отдельного разговора. Ранее мы ввели понятие **потока** – высокопроизводительного состояния человека, в котором он исключительно продуктивно и гармонично применяет свои рабочие инструменты. Состояние потока является естественным, и люди достигают его без внешнего толчка. Требуются определенные усилия, чтобы вывести пользователя из этого состояния. Вмешательство в поток – телефонный звонок или сообщение об ошибке – вполне способно на это. Некоторые из подобных раздражителей неизбежны, но другие вовсе не обязательны.

Прерывание потока без веской причины является *прерыванием работы из-за ерунды*. Это одна из самых разрушительных форм налогового бремени.

Не прерывайте работу из-за ерунды.

Неудачно спроектированный программный продукт позволяет себе делать предположения, которые никогда не сделал бы уважающий себя человек. Бывает, что программа с готовностью выполняет команды пользователя, «подвешивающие» систему так, что требуется перезагрузка компьютера. Пользователи совершенно справедливо расценивают подобное поведение программных продуктов как идиотизм.

Сообщения об ошибках, уведомления и запросы на подтверждение операций

Пожалуй, самой распространенной формой налогов являются сообщения об ошибках и диалоговые окна с требованием подтвердить операцию. Они вездесущи до такой степени, что их искоренение требует большого труда. Эти элементы серьезно обременяют пользователя, и вы должны исключать их из своих приложений, где только возможно.

Типичное диалоговое окно с сообщением об ошибке является абсолют но необязательным. Оно либо сообщает пользователю информацию, до которой ему нет дела, либо требует, чтобы он исправил ситуацию, которую программа обычно может и должна исправить самостоятельно.

Как выявить налоги

Некоторые функции могут оказаться нужными случайному пользователю или пользователю с необычными предпочтениями. Такие функции можно считать налогами только в том случае, если пользователь вынужден выполнять их, не имея альтернативы. Пример такого рода функций – управление размерами и положением окон. Единственный способ определить, является ли та или иная функция налогом, – соотнести ее с целями персонажа. Если важному персонажу необходимо видеть на экране сразу два приложения, чтобы сравнивать или переносить информацию, возможность изменить размеры и расположение их главных окон не является налогом. Если же у персонажа нет такой потребности, необходимость настройки окна любой из этих программ есть не что иное, как налог.

Вы должны неусыпно следить за появлением малейших признаков дополнительных налогов в интерфейсе и изничтожать их на корню. Мириады мелких необязательных операций заставляют пользователя выполнять гигантский объем лишней работы. Следующий список поможет вам выявлять налоги:

1. **Не заставляйте пользователя переходить к другому окну ради выполнения операции, влияющей на текущее окно**.

2. **Не заставляйте пользователя вспоминать, где в файловой иерархии расположены его файлы.**

3. **Не заставляйте пользователя изменять размер окна без необходимости.** Когда на экране появляется дочернее окно, программа должна установить его размер в соответствии с его содержимым. Не создавайте ни большое пустое окно, ни маленькое и требующее про крутки.

4. **Не заставляйте пользователя передвигать окна.** Если на рабочем столе есть свободное место, открывайте окно программы там, а не поверх других окон.

5. **Не заставляйте пользователя заново вводить предпочтения.** Если он установил шрифт, цвет, отступ или громкость, сделайте так, чтобы ему не пришлось повторно выполнять настройку, пока он сам не захочет.

6. **Не заставляйте пользователя вводить данные во все поля в соответствии с неким произвольным критерием полноты.** Если пользователь хочет опустить какие-то детали при вводе данных в форму, не вынуждайте его вводить их. Исходите из предположения, что у пользователя есть веские причины скрывать информацию. Полнота базы данных (в большинстве случаев) не стоит того, чтобы нервировать пользователя.

7. **Не заставляйте пользователя просить разрешения.** Как правило, это признак того, что ввод и вывод отделены друг от друга.

8. **Не просите пользователя подтверждать свои действия** (это правило подразумевает наличие развитой функции отмены).

9. **Не позволяйте действиям пользователя приводить к ошибке.**

Навигация как налог

Самый важный факт, который следует уяснить в отношении навигации, заключается в том, что навигация в основном является налогом. За исключением компьютерных игр, где успешное ориентирование в лабиринте препятствий является *целью*, навигация в программном продукте или на веб-сайте редко отвечает потребностям, целям и желаниям пользователя. (Хотя следует особо отметить ситуацию, когда качественно спроектированная навигация может быть эффективным способом рассказать пользователям о возможностях продукта, что определенно лучше соответствует их целям.)

Необязательная или затрудненная навигация становится серьезным источником раздражения для пользователей. Плохо спроектированная навигация представляет собой *проблему номер один* при разработке любого интерактивного продукта – будь то приложение для работы на настольном компьютере, веб-приложение или что-то иное. Навигация – та составляющая программы, где лучше всего видна модель реализации, использованная программистом.

Будем придерживаться расширенного определения **навигации как любого действия, которое переносит пользователя в новую область интерфейса или требует от него поиска объектов, инструментов либо данных**. Причина этого проста: эти действия требуют, чтобы люди понимали, где находятся в рамках интерактивной системы и как найти и активировать нужную функцию. Когда мы задумываемся об этих действиях как о навигации, становится ясно, что они являются налогом и, следовательно, должны быть исключены или минимизированы.

Навигация в программах происходит на нескольких уровнях:

1. между различными окнами, представлениями или страницами;
2. между панелями или фреймами внутри окна, представления или страницы;
3. между инструментами, командами или меню;
4. по информации, отображаемой внутри панели или фрейма (прокрутка, панорамирование, масштабирование, переход по ссылкам).

Обсудим каждый из этих типов навигации более подробно.

Навигация между экранами, представлениями или страницами

Навигация между несколькими представлениями внутри приложения или веб-страницами является, пожалуй, самым дезориентирующим видом навигации. Навигация между окнами требует переключения внимания, разрушает состояние потока, в котором находится пользователь, и принуждает его перейти в новый контекст. Действие перехода к новому окну нередко приводит к тому, что содержимое исходного окна частично или полностью перекрывается. Пользователю приходится, как минимум, беспокоиться об управлении окнами, а это налог, еще больше разрушающий состояние потока. Если пользователям приходится постоянно переходить от одного окна к другому, продуктивность их работы

падает, а степень дезориентации и раздражения повышается. Они отвлекаются от текущих задач. В монопольных приложениях возникновения этой проблемы можно избежать, поместив основные взаимодействия в одно главное представление, содержащее несколько независимых панелей.

Навигация между панелями

Окно может содержать некоторое количество панелей, соприкасающихся и разграниченных разделителями либо наложенных друг на друга и обозначенных вкладками. Смежные панели могут решить многие навигационные проблемы, поскольку содержат полезные вспомогательные функции, ссылки или данные, имеющие непосредственное отношение к основной рабочей области, что сводит навигацию практически к нулю. Если объекты могут быть перенесены с одной панели на другую, эти панели должны быть расположены по соседству.

Проблемы возникают, когда смежных панелей становится слишком много или когда их расположение на экране не соответствует логической последовательности работы пользователя. Большое количество смежных панелей приводит к визуальной засоренности интерфейса и путанице. Пользователь не знает, где искать необходимое. Кроме то го, переполнение окна панелями вызывает необходимость в прокрутке – а это еще одна навигационная проблема. Тем самым затрудняется навигация в пределах одного окна.

В ряде случаях, в зависимости от решаемых пользователем задач, уместны панели, организованные в виде вкладок. Такая организация создает определенную дополнительную нагрузку и способна дезориентировать пользователя, потому что вкладка, на которую он перешел, закрывает предыдущую. Впрочем, эта идиома уместна в рабочей области, где пользователь имеет дело с несколькими документами или разными внешними представлениями одного документа.

Вкладки – это способ экономить место на экране, применяемый, когда требуется втиснуть всю информацию и функции в ограниченное пространство. Классический пример – диалоговое окно настройки. Не думаем, что комуто захочется увидеть разом все настройки сложного приложения. Однако **в большинстве случаев применение вкладок создает заметные навигационные налоги.** Редко когда удается подобрать достаточно точное название вкладки для панели, с которой она связана, и пользователям приходится просматривать все вкладки в поисках нужной информации или функции.

Вкладки могут быть уместными при наличии нескольких панелей, не используемых одновременно и поддерживающих происходящее в основной рабочей области. Эти вспомогательные панели можно сложить в стопку, и пользователь будет выбирать нужное одним щелчком. Классический пример – микшер цвета и область цветовых образцов в Adobe Illustrator. Эти два инструмента предлагают взаимоисключающие способы выбирать цвет для рисования, и пользователь, как правило, знает, какой способ подходит для текущей задачи.

Навигация между инструментами и меню

Еще одна важная и не осознаваемая разработчиками форма навигации возникает из необходимости пользоваться различными инструментами, палитрами и функциями. Оптимальная пространственная организация соответствующих элементов интерфейса в

пределах панели или окна очень важна для минимизации лишних движений мыши, которые в лучшем случае вызывают у пользователя раздражение и усталость, а в худшем – профессиональные заболевания. **Инструменты, применяемые часто и в сочетании с другими инструментами, следует группировать и делать доступными мгновенно.** Работа с меню требует от пользователя больших усилий по навигации, поскольку содержимое меню невидимо, пока меню не открыто. **Часто применяемые функции должны быть доступны через панели инструментов, палитры или аналогичные элементы интерфейса. Меню должны использоваться только для выполнения команд, к которым пользователь обращается нечасто.**

Навигация по информации

Навигация в информационном содержимом окна или панели осуществляется несколькими методами: прокруткой (панорамированием), переходами по гиперссылкам и масштабированием. Первые два метода широко распространены: прокрутка встречается практически в каждой программе, а переход по ссылкам – практически на каждой веб-странице). Изменение масштаба применяется преимущественно для визуализации объемных и детализированных плоских изображений.

Прокрутка необходима часто, но потребность в ней следует минимизировать. Как правило, удается найти компромисс между разбивкой на страницы и прокруткой информации. Вы должны понять ментальные модели ваших пользователей и последовательность действий в их работе, чтобы выбрать подходящий вариант.

В двухмерных графических программах вертикальная и горизонтальная прокрутка в порядке вещей. Навигация еще больше упрощается при наличии миникарты. Эта техника обсуждается ниже в данной главе наряду с другими визуальными указателями.

Гиперссылки – очень важная навигационная парадигма в среде Все мирной паутины. Поскольку она изменяет внешний вид экрана, разработчик должен позаботиться о визуальных и текстовых навигационных ориентирах для пользователя.

Масштабирование и **панорамирование** – это навигационные инструменты для работы с двухмерными и трехмерными изображениями. Они уместны при создании двухмерных и трехмерных сцен и моделей и при просмотре трехмерных моделей реального пространства (напри мер, в виртуальных экскурсиях). Они, как правило, не способны оказать помощь при исследовании случайных или абстрактных данных, представленных более чем в двух измерениях. В некоторых программах визуализации масштабирование означает более детализированный показ объекта, то есть имеет скорее логический, чем пространственный характер. По мере увеличения объекта его атрибуты (нередко текстовые) появляются поверх изображения. Такой вид взаимодействия обычно дает хорошие результаты, если реализован через смежную вспомогательную панель, представляющую свойства выделенных объектов в стандартизированной хорошо читаемой форме. **Пользователям трудно разобраться в пространственном масштабировании; логическое масштабирование является загадочным для всех, кроме профессионалов в сфере визуализации и редких программистов.**

Панорамирование и масштабирование, а тем более их сочетание создают огромные навигационные проблемы для пользователей. И хотя ситуация улучшается с развитием

доступных через Интернет карт, пользователю попрежнему легко заблудиться в виртуальной реальности.

Люди не привыкли перемещаться в неограниченном трехмерном пространстве, и они с трудом воспринимают трехмерное пространство, спроецированное на двухмерный экран.

Улучшение навигации

Существует много способов улучшения (исключения, сокращения, ускорения) навигации по вашим программам, вебсайтам и электронным устройствам. Вот наиболее эффективные:

1. уменьшение количества пунктов назначения;
2. создание «дорожных указателей»;
3. организация обзора;
4. ассоциирование элементов управления с функциями;
5. адаптация интерфейса к нуждам пользователя;
6. отказ от иерархических структур.

Обсудим эти методы более подробно.

Уменьшение количества пунктов назначения

Самый эффективный метод улучшения навигации очевиден: **сократите число пунктов назначения при работе с продуктом.** Под «пунктами назначения» здесь понимаются рабочие режимы, формы, диалоговые окна, страницы, окна и экраны. Когда количество режимов, страниц или экранов минимально, возможности пользователя по ориентированию в программе значительно возрастают. В терминах четырех типов навигации, перечисленных ранее, это означает следующее:

1. **Сведите количество окон и представлений к минимуму.** Одно полноэкранное окно с двумя, максимум тремя представлениями – оптимальный вариант для многих пользователей. Сведите к минимуму количество диалоговых окон. Программы и вебсайты с десятками различных типов страниц, экранов и форм сложны для навигации.
2. **Сократите количество смежных панелей в окне или на веб-странице до минимального количества, необходимого пользователю для достижения своих целей.** Разумный максимум для монопольных приложений – не более трех панелей, однако здесь нет жестких правил: многим приложениям требуется больше панелей. На веб страницах все, что сложнее двух областей навигации и одной области с данными, вызывает у пользователей излишнее напряжение.

3. **Сведите количество элементов управления к тому минимуму, который позволяет пользователям достигать своих целей.** Понимание пользователей, которое дают персонажи, позволит вам исключить функции и элементы управления, о которые пользователи будут спотыкаться ввиду их ненужности или нежелательности.

4. **Минимизируйте прокрутку везде, где это возможно.** Иными словами, создавайте вспомогательные панели такого размера, чтобы информация, которую они содержат, не нуждалась в постоянной прокрутке. Двухмерные и трехмерные графики и сцены должны по умолчанию иметь такой размер, чтобы пользователю не приходилось постоянно прибегать к панорамированию. Масштабирование, особенно когда оно выполняется неоднократно, является для большинства пользователей самым трудным типом навигации, так что его применение должно быть вызвано желанием пользователя, а не суровой необходимостью.

Создание «дорожных указателей»

Помимо сокращения числа пунктов назначения можно дополнительно упростить пользователям ориентирование при помощи более качественных подсказок – **дорожных указателей**. Подобно морякам, ориентирующимся по звездам и береговой линии, **пользователи ориентируются по стабильным объектам, встроенным в интерфейс.**

На рабочем столе стабильными объектами являются окна программ. Каждая программа, как правило, имеет главное окно (окно верхнего уровня). Самые заметные особенности главного окна тоже могут считаться стабильными объектами – это строки меню, панели инструментов и прочие палитры и визуальные элементы, такие как строки состояния и линейки. Вообще говоря, **каждое окно интерфейса имеет свои отличительные черты и быстро становится узнаваемым.**

В среде Всемирной паутины действуют аналогичные правила. **В качественно спроектированные веб-сайты тщательно внедрены стабильные объекты, которые не изменяются в процессе навигации**. Это, в частности, меню навигации вверху страницы. Такие области не только четко передают навигационные возможности, но и своим постоянным присутствием на экране помогают пользователям ориентироваться.

Для успешной навигации не требуется большого количества дорожных указателей. Достаточно, чтобы они были хорошо видны. Нет нужды говорить, что указатели, которые вдруг пропадают, не способствуют навигации. Это означает, что они должны быть постоянной принадлежностью интерфейса.

Стремление сделать все страницы вебсайта похожими друг на друга продиктовано маркетинговыми соображениями, однако, если зайти слишком далеко, пользователей это может и дезориентировать. Ясно, что общие элементы страниц должны выглядеть и располагаться единообразно. Тем не менее **если разделы сайта будут визуально различаться, это поможет пользователям ориентироваться**.

Организация обзора

Обзоры играют в интерфейсе ту же роль, что и дорожные указатели: они способствуют ориентированию пользователей. Разница между ними заключается в том, что обзоры

помогают пользователю ориентироваться внутри информационного содержания, а не в приложении как таковом. Поэтому сама область с обзором должна быть стабильным объектом; содержание же ее зависит от просматриваемых данных.

В зависимости от характера информации обзоры могут быть графическими или текстовыми. Блестящим примером графического обзора является панель с уместным названием Navigator в Adobe Photoshop.

Во Всемирной паутине самой распространенной формой обзора является текст, а именно – вездесущие «хлебные крошки». Они не только показывают пользователю, в какой точке иерархической структуры данных он находится, но и снабжают его ссылками, позволяющими переходить к другим узлам этой структуры. Эта идиома до некоторой степени утратила популярность, когда веб-сайты стали отходить от иерархической организации в пользу ассоциативной организации, которая не слишком хорошо уживается с «хлебными крошками».

Интересным инструментом обзора является **аннотированная полоса прокрутки**. Такие полосы особенно уместны при прокрутке текста. Они грамотно используют тот факт, что как полосы прокрутки, так и текстовая информация линейны по своей природе. Это позволяет выводить на экран информацию о местонахождении выделенных областей и, воз можно, других атрибутов форматированного или неформатированного текста. Подсказки о том, где находятся эти элементы, появляются в соответствующие моменты на траектории движения ползунка прокрутки.

Ассоциирование элементов управления с функциями

Ассоциирование описывает отношение между элементом управления, объектом воздействия и получаемым результатом. Некачественное ассоциирование проявляет себя в том, что элемент управления не соотносится ни визуально, ни символически с объектом, на который он действует. Некачественная ассоциация заставляет пользователя остановиться и задуматься о соотношениях, а это выводит его из состояния потока. Плохое ассоциирование элементов управления с функциями увеличивает когнитивную нагрузку на пользователя и чревато серьезными ошибками.

Хороший пример проблем, связанных с ассоциированием, мы наблюдаем в интерфейсе обычной кухонной плиты, далекой от мира компьютерных технологий. Практически каждый, кому приходилось готовить, испытывал раздражение от неподходящего ассоциирования ручек кухонной плиты с горелками, которые они открывают. У типичной плиты горелки расположены по углам квадрата. Однако ручки для этих горелок расположены в ряд на передней панели плиты.

В этом случае мы имеем дело с проблемой **физического ассоциирования**. *Результат* использования элемента управления достаточно очевиден: горелка включится, когда вы повернете ручку. Однако неясен *целевой объект* элемента управления: какая именно горелка нагреется. Пользователи должны выяснять это либо методом тыка, либо разглядывая крохотные пиктограммы рядом с ручками. Неестественность ассоциации вынуждает пользователей каждый раз изучать ее заново. Это действие со временем уходит в подсознание, но оно выполняется каждый раз и может закончиться ошибкой, когда пользователь спешит или отвлекается (что нередко происходит во время приготовления еды). В лучшем случае пользователь почувствует себя глупо, если повернет не ту ручку, а еда останется холодной, пока он не заметит ошибку. В худшем дело может закончиться ожогами и пожаром.

Решение этой проблемы заключается в таком изменении расположения ручек, чтобы было понятно, какой горелкой управляет каждая из них. Совсем не обязательно располагать ручки в точности так же, как горелки, однако их позиции должны ясно показывать целевую горел ку каждой ручки.

Создаете ли вы бытовые устройства, настольные приложения или же вебсайты, вы всегда можете столкнуться с проблемами ассоциирования. Вы можете существенно улучшить свой продукт за счет поиска и устранения проблем, связанных с ассоциированием, даже если у вас мало времени на внесение изменений. Что в результате? Продукт, в котором легче разобраться и с которым приятнее иметь дело.

Адаптация интерфейса к нуждам пользователя

Адаптация интерфейса подразумевает организацию его таким образом, чтобы минимизировать объем типичной навигации. На практике это означает **размещение наиболее востребованных функций и элементов управления в самых удобных для пользователя местах и перенос редко используемых функций вглубь интерфейса**, чтобы пользователь не «спотыкался» о них. Не следует удалять из программы редко используемые функциональные возможности, однако убрать их из рабочей области пользователя необходимо.

Самым важным принципом правильной адаптации интерфейса является принцип **соразмерности усилий**. Хотя он действует в отношении всех пользователей, он особенно уместен, когда речь идет о средних пользователях. Это принцип гласит, что **люди готовы прилагать дополнительные усилия, если результат того стоит.** Естественно, ценность результата определяется пользователем. Она не имеет никакого отношения к тому, насколько трудно было реализовать функциональную возможность, и полностью зависит от целей пользователя. Сказанное означает, что если вы добавите к программе функциональные возможности, которыми сложно пользоваться, пользователи будут готовы терпеть трудности только в обмен на достойное вознаграждение. Поэтому пользовательский интерфейс вашей программы не должен быть сложным, когда речь идет о достижении простых результатов, но *может* быть сложным при необходимости достигать *сложных* результатов (и при условии, что такая необходимость возникает не слишком часто).

Элементы управления и окна должны быть организованы в интерфейсе по трем параметрам – частоте использования, степени влияния на внешний вид интерфейса и степени риска.

1. **Частота использования** определяет, как часто пользователь обращается к элементам управления, функциям, объектам и окнам при повседневном использовании продукта. Элементы и инструменты, необходимость в которых возникает чаще всего (много раз в течение дня), должны всегда быть под рукой. Элементы, используемые реже (один или два раза в день), должны быть не дальше, чем «на расстоянии» одногодвух щелчков. Доступ к прочим элементам может потребовать двух или трех щелчков.

2. **Степень влияния на внешний вид интерфейса** означает масштаб резких изменений в интерфейсе или обрабатываемом документе (информации), вызываемых активизацией той или иной функции либо команды. Вообще говоря, команды, оказывающие сильное влияние, следует прятать подальше.

3. **Степень риска** относится к функциям, выполняющим необратимые действия или имеющим другие опасные последствия. Инструкции по запуску межконтинентальных

баллистических ракет требуют, чтобы два человека одновременно повернули ключи в противоположных концах помещения – иначе запуск невозможен. Как и в случае с функциями, изменяющими внешний вид интерфейса, опасные функции следует убрать подальше, чтобы пользователи на них не натыкались. Уровень риска можно считать произведением вероятности события на нежелательные последствия этого события.

Отказ от иерархических структур

Иерархические структуры – один из самых надежных инструментов программиста. Большая часть данных внутри программы, да и сам ее код имеют иерархическую организацию. По этой причине многие программисты применяют иерархии (модель реализации) и в пользовательских интерфейсах. Как мы помним, первые меню были иерархическими. Однако пользователям трудно осуществлять навигацию в абстрактных иерархических структурах, за исключением тех случаев, когда эти иерархии основаны на ментальных моделях пользователей, а категории являются действительно взаимоисключающими. Программисты часто отказываются принять этот факт, поскольку чувствуют себя рядом с иерархическими структурами вполне комфортно.

Большинство механических систем хранения информации просты и состоят либо из одной последовательности объектов (как книжная полка), либо из нескольких последовательностей, имеющих один уровень глубины (как в картотечном шкафу с папками). Такой способ организации, подразумевающий один слой из не скольких групп, очень распространен, и его можно встретить в каждом доме и офисе.

Программисты легко работают с системами многоуровневой вложенности, где экземпляр какого-либо объекта хранится в другом экземпляре того же объекта. Большинству непрограммистов очень трудно воспринять эту идею. В сложных механических системах хранения информации на разных уровнях неизбежно используются разные методы хранения. Вы никогда не найдете папки внутри папок в картотечном шкафу или ящики внутри ящиков. Даже неоднородная схема «папка – ящик – шкаф» редко превышает два уровня вложенности. В метафоре рабочего стола, применяемой большинством оконных систем, вы можете вкладывать папки в папки до бесконечности.

Вместо того чтобы вынуждать пользователей осуществлять навигацию по многоуровневым древовидным структурам, дайте им инструменты для *удобного доступа к нужной информации*.

Проектирование хорошего поведения

Если мы хотим, чтобы пользователям понравился наш программный продукт, он должен вести себя так, как ведет себя человек, приятный в общении. Если мы хотим, чтобы работа пользователя с нашей программой была продуктивна, программа должна вести себя как товарищ по работе, готовый поддержать коллегу. Чтобы этого добиться, полезно рассмотреть уместные рабочие отношения между людьми и компьютерами.

Идеальное разделение труда в век компьютеров таково: компьютер работает, а человек думает. Писатели-фантасты и специалисты в области компьютерных технологий дразнят нас перспективами развития искусственного интеллекта: компьютеры будут думать за себя сами. Однако пользователям не особенно требуется помощь в мыслительных процессах: наша способность выявлять закономерности и творчески решать сложные задачи не имеет аналогов в мире кремния. Что нам действительно *требуется*, так это помощь в управлении информацией – в такой деятельности, как доступ, анализ, организация и визуализация информации. А принятие решений на основе полученной информации лучше всего получается у людей.

Проектирование тактичных продуктов

Программный продукт должен быть *тактичным*. Подлинная *тактичность* означает первоочередное внимание к интересам других. Тактичное программное обеспечение ориентировано в первую очередь на цели и нужды пользователей, а не на собственные базовые функции.

Если программа скупа на информацию, окружает свои процессы ореолом тайны, заставляет прилагать усилия для поиска обычных функций и с легкостью обвиняет пользователя в своих промахах, человеку обеспечен неприятный и непродуктивный опыт.

Зато продукт, который ведет себя уважительно и благородно и готов прийти на помощь пользователю, делает огромный шаг к созданию положительного опыта у работающих с ним людей.

Программный продукт должен вести себя как тактичный человек.

Интерактивные продукты часто раздражают нас потому, что они бестактны, а не потому, что им не хватает какихто возможностей. Совершенно не факт, что создание тактичного продукта значительно сложнее, чем создание грубого или бестактного. Нужно всего-навсего представить себе взаимодействие, которое имитирует качества чувствительного и заботливого человека. Ни одно из этих качеств не противоречит более прагматичным целям обработки данных, стоящим перед любым цифровым продуктом. Сверх того, более гуманное поведение может оказаться самой прагматичной из всех целей, и в сочетании с правильной оркестровкой подобный диалог с пользователями может внести большой вклад в функциональность продукта.

У людей есть множество прекрасных качеств, которые делают их тактичными, и некоторые из них можно в той или иной степени воспроизводить в интерактивных продуктах. Следующий перечень содержит некоторые наиболее важные качества тактичных интерактивных продуктов (и людей):

1. Проявлять интерес

2. Вести себя почтительно

3. Проявлять услужливость
4. Проявлять здравый смысл
5. Предупреждать желания людей
6. Проявлять инициативность
7. Не перекладывать на других свои проблемы
8. Держать коллегу в курсе дела
9. Проявлять понятливость
10. Иметь уверенность в себе
11. Не задавать лишних вопросов
12. Принимать ответственность
13. Знать, когда можно отклониться от правил

Рассмотрим эти качества подробнее.

Тактичные продукты проявляют интерес к людям

Хороший друг хочет знать о вас больше. Он запоминает, что вам нравится, а что не нравится, чтобы сделать вам чтонибудь приятное в будущем. Любому человеку нравится, когда кто-то учитывает его вкусы. С другой стороны, большинству программ безразлично, кто ими пользуется. На наших *персональных* компьютерах практически нет программных продуктов, которые знали бы о нас хоть чтонибудь *личное*, хотя мы используем их постоянно, многократно и безальтернативно. Положительный пример такого поведения демонстрируют броузеры Firefox и Microsoft Internet Explorer, которые запоминают вводимую пользователем на сайтах информацию, например адреса доставки или регистрационное имя.
Программа должна изо всех сил стараться запомнить привычки пользователя и особенно команды, которые поступали от него. Программисту, который ее создал, эта информация кажется сиюминутной – когда она нужна программе, программа просто требует, чтобы пользователь сообщил необходимые сведения, а затем забывает их, полагая, что за просит их снова в случае необходимости. Несмотря на то, что программа способна запоминать информацию лучше, чем человек, она все за бывает, проявляя *бестактность*. Запоминание действий и предпочтений людей – это один из лучших способов пробудить положительные эмоции от применения цифрового продукта. Мы еще вернемся к теме памяти далее в этой главе.

Тактичные продукты ведут себя почтительно

Хороший работник сферы обслуживания проявляет почтительность по отношению к клиенту. Он относится к клиенту, как к начальнику.

Когда метрдотель в ресторане показывает нам, за какой столик сесть, мы воспринимаем его выбор столика как предложение, а не как приказ. Если мы вежливо попросим его предоставить нам другой столик, когда ресторан почти пуст, то ожидаем, что нас разместят за другим столиком. Если метрдотель откажет нам, мы, скорее всего, предпочтем другой ресторан, где *наши* пожелания ставятся выше пожеланий метрдотеля.

Нетактичный продукт ведет себя как начальник и осуждает поступки человека. Программа имеет право высказывать *мнение*, что мы ошиблись, однако судить наши действия – это дерзость с ее стороны. Программа может *предположить*, что лучше не щелкать по кнопке Submit, пока мы не укажем номер своего телефона, и объяснить возможные по следствия, но, если мы захотим отправить данные без номера телефона, мы вправе ожидать, что программа поступит так, как велено.

Тактичные продукты услужливы

Если мы спросим работника универсама, как найти необходимый нам товар, он не только ответит на наш вопрос, но и по своей инициативе сообщит более важную сопутствующую информацию – например, что более дорогой и качественный аналогичный товар продается сейчас со скидкой почти за такую же цену.

Большинство программ даже и не пытаются предоставить сопутствующую информацию. Они дают ограниченный ответ на конкретный вопрос, не заботясь о предоставлении других сведений, даже если те непосредственно связаны с нашими целями. Когда мы выдаем текстовому редактору команду распечатать документ, он не сообщает нам, что в лотке принтера мало бумаги, что перед нами очередь из сорока документов, или что другой стоящий неподалеку принтер свободен. Дружелюбный коллега сообщил бы нам об этом.

Поиск правильного способа предлагать потенциально полезную информацию может оказаться непростым. Скрепыша, созданного компанией Microsoft, презирают практически все – за ее «умные» комментарии вроде этого: «Похоже, вы набираете письмо. Моя помощь не требуется?» Настрой у Скрепыша похвальный, но хотелось бы, чтобы он вел себя не столь нагло и понимал, когда пользователь явно не желает его видеть. В конце концов, хороший официант не спрашивает клиента, хочет ли тот еще воды, а просто наполняет опустевший стакан, и имеет достаточно мозгов, чтобы не шнырять вокруг да около, если видит, что клиент увлечен беседой с девушкой.

Тактичные продукты проявляют здравый смысл

Неподходящие функции в неподходящих местах – отличительный признак плохо спроектированных интерактивных продуктов. В большинстве из них элементы управления востребованными функциями расположены рядом с элементами, которые не используются никогда. Вы легко найдете программное меню, где простые безвредные функции соседствуют с функциями, имеющими необратимые последствия («рычаги катапульты») и предназначенными исключительно для экспертов.

Тактичные продукты предупреждают потребности человека

Секретарша знает, что вам потребуется номер в гостинице, когда вы отправляетесь в командировку в другой город, даже если вы не попросили ее заказать номер. Она знает, какую комнату вы предпочитаете, и закажет номер без напоминаний с вашей стороны. Она предупреждает ваши потребности.

Веб-браузер большую часть времени простаивает, пока вы просматриваете веб-страницы. Он прекрасно мог бы предупредить ваши потребности и подготовиться к ним. Например, он бы мог за это время загрузить страницы, ссылки на которые представлены на экране. Велика вероятность, что мы захотим перейти по каким-нибудь из этих ссылок. Отменить нежелательный запрос легко, а ждать его выполнения долго.

Тактичные продукты инициативны

Инициативный человек демонстрирует более широкий взгляд на порученное дело. Например, он не только помоет посуду, но и вытрет стол и выбросит окурки из пепельницы, поскольку эти действия тоже служат глобальной *цели*: навести порядок на кухне. Когда инициативный работник пишет отчет, он не забудет о красивой обложке и сделает достаточное количество ксерокопий, чтобы хватило всем заинтересованным лицам.

Рассмотрим пример. Вы вручаете своему помощнику папку с документами и велите убрать на место. Он читает надпись на папке, скажем «Контракты с MicroBlitz», и смотрит, куда ее поставить в картотечном шкафу. Под буквой М он, к своему удивлению, находит папку с точно такой же надписью. Заглянув в нее, он выясняет, что она содержит контракт на 17 изделий, поставленных фирме MicroBlitz четыре месяца назад. Зато в новой папке находится контракт на 32 шестеренки, изготовление и поставка которых запланированы на следующий квартал. Ваш инициативный помощник приклеивает к старой папке яр лык «Контракт с MicroBlitz 7/03», а к новой – ярлык «Контракт с MicroBlitz на шестеренки 11/03». Именно такая инициатива позволяет нам считать его инициативным сотрудником.

Предыдущий ваш помощник был совершенным болваном. К тому же он не был инициативным и, окажись он в этой ситуации, он без тени сомнения поставил бы новую папку рядом со старой. Конечно, он выполнил бы ваше поручение, но мог бы сделать это лучше, что в будущем сократило бы время поиска нужных документов. Потому-то этот человек у вас больше не работает.

Если мы создадим в текстовом редакторе документ с контрактом на шестеренки и попытаемся сохранить его в каталоге MicroBlitz, то редактор предложит нам либо уничтожить старый контракт, либо отказаться от сохранения нового. Программа не только не обладает способностями вашего теперешнего помощника – она даже глупее предыдущего, который был законченным болваном. Она настолько тупа, что усматривает в ваших действиях желание выбросить старый документ лишь потому, что новому выдали то же название.

Программа, как минимум, должна пометить файлы разными датами и сохранить их. Даже если она не способна выполнить это «радикальное» действие в одностороннем порядке, она должна хотя бы показать вам старый файл (и позволить переименовать его), прежде чем сохранять новый. Существует масса действий, которые программа могла бы выполнить, будь она более инициативной.

Тактичные продукты не перекладывают на вас свои личные проблемы

Общаясь с клиентом, агент любой фирмы помалкивает о своих личных проблемах и выказывает разумный интерес к проблемам клиента. Возможно, подобная однобокость не очень справедлива, но такова суть сферы обслуживания. Точно так же интерактивный продукт должен помалкивать о своих проблемах и проявлять интерес к проблемам использующих его людей. Поскольку компьютеры не имеют своего «я» и не являются чувствительными особами, они превосходно подходят для этой роли. Тем не менее они, как правило, ведут себя прямо противоположным образом.

Программа выдает сообщения об ошибках, прерывает нашу работу диалоговыми окнами, требующими подтверждения, и хвастается, выводя на экран никому не нужные уведомления. («Документ успешно сохранен!») Нам нет дела до приступов неуверенности в себе, случающихся у программы при необходимости очистить Корзину. Мы не желаем слушать ее нытье о том, куда помещать файл. Информация о скорости передачи данных и о последовательности загрузки интересует нас не больше, чем рассказ о несчастной любви представителя службы поддержки. Программа не только не должна распространяться о своих проблемах – у нее должно быть достаточно интеллекта, уверенности и полномочий, чтобы решить эти проблемы самостоятельно.

Тактичные продукты держат нас в курсе дел

Хотя мы и не желаем, чтобы программа непрестанно докучала нам со общениями о своих страхах и маленьких победах, мы хотим быть в курсе событий, важных для *нас*. Мы не желаем выслушивать рассказы бармена о его недавнем разводе, но будем ему благодарны, если он поместит ценники на видном месте и повесит объявление о том, когда будет трансляция ближайшего футбольного матча, кто играет и сколько очков у нашей любимой команды. Эта информация не прерывает нашу деятельность, она просто находится в поле зрения. Подобно этому программа может обеспечить насыщенную обратную связь, сообщая о том, что происходит.

Тактичные продукты понятливы

Большинство существующих программных продуктов не слишком сообразительны. В большинстве случаев программы демонстрируют очень узкое понимание проблемы. Программа хорошо справляется с трудной работой, но лишь в том случае, когда получает четкую команду в нужный момент времени. Если, например, вы попросите систему автоматизированного управления складом сообщить, каков запас каких-нибудь деталей, она послушно сверится с базой данных и выдаст вам точное количество деталей на момент запроса. А что, если через двадцать минут кто-то в иногороднем филиале вашей фирмы закажет со склада весь запас этих деталей? Вы будете принимать решения, находясь в заблуждении, а ваш компьютер будет стоять без дела, пропуская миллиарды циклов. Это неумно с его стороны. Если вы однажды запросили информацию о деталях, разве это не должно навести на мысль, что вы запросите информацию о них снова? Возможно, вы не захотите получать отчеты о наличии деталей ежедневно до конца жизни, но не исключено, что они будут нужны вам в течение недели. Понятливая программа следит за действиями пользователя и на их основании предлагает ему соответствующую информацию.

Продуктам следует внимательно относиться и к нашим предпочтениям, запоминать их самостоятельно, без подсказки. Если пользователь всегда разворачивает окно в полный экран, приложение после нескольких сеансов должно запомнить это и всегда запускаться с уже развернутым окном. То же самое относится к расположению палитр, стандартных инструментов, часто используемых шаблонов и прочим полезным настройкам рабочей среды.

Тактичные продукты уверены в себе

Интерактивные продукты должны вести себя уверенно. Если мы сообщаем компьютеру, что нужно удалить файл, он не должен спрашивать: «Вы уверены?» Конечно, мы уверены, иначе бы не попросили об этом. Он не должен перепроверять нас или себя.

С другой стороны, если у компьютера возникает подозрение, что мы ошибаемся (а оно у него присутствует постоянно), он должен предвидеть, что мы передумаем, и быть готовым восстановить файл по нашей просьбе.

Как часто вам доводилось щелкнуть по кнопке Печать и уйти пить кофе, а потом, вернувшись, увидеть в центре экрана ужасное диалоговое окно с вопросом:«Вы уверены, что хотите отправить документ на печать?» Невозможность положиться на программу способна привести в бешенство и является полной противоположностью тому, как мы представляем себе тактичное поведение человека.

Тактичные продукты не задают лишних вопросов

Нетактичная программа задает массу раздражающих вопросов. Слишком широкий выбор быстро перестает быть достоинством и превращается в пытку.

Варианты выбора можно предлагать поразному. Например, их можно выложить, как товары в витрине магазина. Мы смотрим на витрину, когда нам заблагорассудится, разглядываем, выбираем или игнорируем предложенные товары. Никто не задает нам вопросов. И, наоборот, выбор можно навязывать пользователю, как вопросы таможенника на границе: «Вы провозите чтонибудь, что надо включить в декларацию?» Мы не знаем последствий этого вопроса. Обыщут нас или нет? Программа никогда не должна устрашать пользователей подобным образом.

Тактичные продукты аккуратно обрабатывают свои сбои

Когда ваш друг совершает проступок, он после этого старается исправить то, что еще может быть исправлено. Когда программа обнаруживает фатальную проблему, она оказывается перед выбором – либо попытаться подготовиться к провалу и не причинить вреда пользователю, либо «разбиться и сгореть».

Большинство программ имеют дело с большими объемами данных и множеством параметров. При аварийном завершении работы эта информация, как правило, безвозвратно теряется, и пользователь остается на перроне с чемоданом в руках. Например, программа принимает электронное письмо на ваше имя, и ей не хватает памяти во время выполнения какой-то процедуры, спрятанной глубоко в ее недрах. Как типичное настольное приложение она выдает

сообщение, в конечном счете означающее: «Вы облиты с ног до головы», – и закрывается, как только вы щелкнете по кнопке OK. Вы перезапускаете программу (а иногда и перезагружаете компьютер) и обнаруживаете, что программа потеряла письмо. А связавшись с почтовым сервером, вы узнаете, что он стер письмо, поскольку оно было передано вашей программе. Качественная программа не должна вести себя подобным образом.

В приведенном примере программа приняла письмо от сервера, который стер копию, но не позаботилась о том, чтобы письмо было надлежащим образом сохранено на вашем компьютере. Если бы программа убедилась, что письмо сохранено на локальном диске, до того, как она сообщила серверу об успешном получении письма, проблема бы не возникла.

Тактичные продукты знают, когда можно отклониться от правил

Когда системы ручной обработки информации подвергаются компьютеризации, чтото неизбежно теряется. Хотя системы автоматизированной обработки заказов могут принять на миллион заказов больше, чем клерк, клерк способен *обойти инструкции*, что немыслимо для автоматизированных систем. В автоматизированной системе почти никогда нет способа отойти от заданного образа действий ради некоторого выигрыша.

Если при ручной обработке заказов клерку позвонит его приятель из отдела продаж и объяснит, что быстрая обработка того или иного заказа может обернуться выгодной сделкой, клерк ускорит свою работу над данным заказом. Если в какомто заказе будет отсутствовать необходимая информация, клерк частично обработает его и не забудет за просить эту информацию впоследствии, чтобы внести ее в заказ. Автоматизированные системы, как правило, не могут похвастать подобной гибкостью.

Большинство автоматизированных систем признают только два со стояния: объект либо отсутствует, либо полностью соответствует требованиям. Промежуточные состояния не распознаются и не принимаются. В любой системе ручной обработки возможен важный, хотя и парадоксальный вариант – неоговоренный, недокументированный, но широко принятый: объект может находиться в **подвешенном** со стоянии, когда транзакция принята, но все еще не доведена до конца. Клерк создает это состояние в уме, или на рабочем столе, или в заднем кармане.

Пусть, например, цифровой системе требуется информация о клиенте и о заказе, чтобы выписать счет. В то время как клерк может выписать счет до того, как получит информацию о клиенте, компьютерная сис тема отвергнет транзакцию, не желая выписывать счет без полных сведений.

Характеристика ручных систем обработки, позволяющая работникам выполнять действия в нестандартной последовательности или до того, как будут соблюдены все формальности, называется **сговорчивостью**. Сговорчивость становится одной из первой жертв компьютеризации, а отсутствие этого качества является основной причиной «нечеловечности» компьютерных систем. Налицо естественный результат следования модели реализации. Программист не видит необходимости в создании промежуточных состояний, потому что компьютер в них не нуждается. Тем не менее существует сильная человеческая потребность – иметь возможность слегка отступать от правил.

Одним из достоинств сговорчивых систем является уменьшение количества ошибок. Допуская наличие в системе множества мелких временных ошибок и предполагая, что люди

исправят их до того, как возникнут проблемы на следующих этапах работы, мы сможем избежать более серьезных ошибок. Парадоксально, но большинство строгих правил, сформулированных в компьютерных системах, имеют целью предотвращение как раз таких ошибок. Эти негибкие правила разделяют людей и программы на два враждующих лагеря, и, поскольку людям не разрешается допускать недоделки ради предотвращения крупных ошибок, они не заботятся о защите программ от действитель но колоссальных проблем. Когда негибкие ограничения накладывают ся на людей, которым свойственна гибкость, проигрывают обе стороны. В конечном итоге, если людям не позволяется делать то, что они хотят, страдает бизнес, а компьютерным системам все равно в конце концов приходится обрабатывать неверные данные.

В реальном мире недостающая информация (равно как излишняя ин формация, не вписывающаяся в стандартный формат) нередко являет ся важным средством на пути к успеху. Системы обработки информации редко способны управиться с данными из реального мира. Они моделируют лишь жесткие повторяющиеся данные из транзакций, что то вроде скелета реальных транзакций, которые включают в себя встречи с людьми, путешествия и развлечения, игру в гольф, имена супругов, детей и звезд спорта. Бывает, что транзакция может быть завершена только через две недели после установленного официального срока. Большинство компаний предпочтет принять работу с недодел ками, когда подойдет срок, чем наблюдать, как многомиллионная сделка превращается в дым. В реальном мире ограничения нарушают ся постоянно. Тактичный продукт должен осознавать и учитывать этот факт.

Тактичные продукты берут на себя ответственность

Слишком многие интерактивные продукты занимают позицию «я за это не отвечаю». Когда такая программа передает поручение аппарат ной части устройства, она «умывает руки», полагая, что глупая аппаратура сама закончит дело. Любому пользователю ясно, что такая про грамма не является ни тактичной, ни ответственной, что она не подставляет плечо под общую ношу и не помогает пользователю работать эффективнее.

Рассмотрим типичный пример – печать. Приложение отправляет от чет на 20 страниц принтеру и одновременно выводит диалоговое окно наблюдения за процессом с кнопкой отмены. Если пользователь быстро сообразит, что забыл внести одно важное изменение, он щелкнет по кнопке, как только первая страница показалась из принтера. Про грамма тут же отменит операцию печати. Однако без ведома пользователя, пока принтер готовился выводить первую страницу, компьютер успел послать 15 страниц в буфер принтера. Программа отменяет печать последних пяти страниц, но принтер об отмене не знает ровным счетом ничего. Он знает про первые 15 страниц и продолжает печатать их. Тем временем программа самодовольно сообщает пользователю, что операция отменена. Программа лжет – и пользователь видит это.

Пользователю нет дела до проблем со связью между приложением и принтером. Его не волнует, что эта связь односторонняя. Зато он твердо знает, что передумал печатать документ до того, как первая страница появилась в выходном лотке, знает, что он щелкнул по кнопке Отмена и что после этого глупая программа продолжила печатать и вывела 15 страниц, хотя он заблаговременно приложил все усилия, чтобы прекратить печать. Причем программа ведь даже подтвердила команду отмены! Выбросив 15 испорченных страниц в мусорную корзину, пользователь недовольно рычит на глупую программу.

Представьте, как развивались бы события, если бы программа могла обмениваться сообщениями с драйвером принтера, а тот – с принтером. Если бы программа была достаточно интеллектуальна, задание на печать можно было бы без труда отменить еще до того, как была испорчена вторая страница. У принтера определенно есть функция отмены, просто программа слишком ленива, чтобы воспользоваться ею.