Критерии качества интерфейса пользователя

Существует четыре основных критерия качества любого интерфейса, а именно:

- скорость работы пользователей,
- количество человеческих ошибок,
- скорость обучения,
- субъективное удовлетворение пользователей (подразумевается, что соответствие интерфейса задачам пользователя является неотъемлемым свойством интерфейса).

1. Скорость выполнения работы

Скорость выполнения работы является важным критерием эффективности интерфейса. В чистом виде этот критерий ценят довольно редко, но почти всегда он является крайне желательной составляющей целого. Любая попытка как-то увеличить производительность труда всегда встречается положительно.

Длительность выполнения работы пользователем состоит из следующих составных частей:

- длительности восприятия исходной информации;
- длительности интеллектуальной работы (в смысле – пользователь думает, что он должен сделать);
- длительности физических действий пользователя;
- длительности реакции системы.

Длительность восприятия исходной информации в особых комментариях не нуждается, Пользователь должен представить себе, какая информация о выполняемой задаче у него существует, и в каком состоянии находятся средства, с помощью которых он будет решать данную задачу. Основное время здесь пойдет на считывание показаний системы.

Длительность интеллектуальной работы оценивается тем, что взаимодействие пользователя с системой (не только компьютерной) состоит из семи шагов:

1. Формирование цели действий.
2. Определение общей направленности действий.
3. Определение конкретных действий.
4. Выполнение действий.
5. Восприятие нового состояния системы.
6. Интерпретация состояния системы
7. Оценка результата.

Из этого списка становится видно, что процесс размышления занимает почти все время, в течение которого пользователь работает с компьютером, во всяком случае, шесть из семи этапов полностью заняты умственной деятельностью. Соответственно, повышение скорости этих размышлений приводит к существенному улучшению скорости работы.

К сожалению, существенно повысить скорость собственно мышления пользователей невозможно. Тем не менее, уменьшить влияние факторов, усложняющих и, соответственно, замедляющих процесс мышления, вполне возможно. Например, применением метода непосредственного манипулирования и применения в интерфейсе эффективных методов при потере фокуса внимания.

Непосредственное манипулирование

Смысл этого метода очень прост. Пользователь не отдает команды системе, а манипулирует объектами. Когда вы хотите зажечь газ в плите, вы ведь не командуете плите «Зажги газ!». Нет, вы манипулируете спичками и плитой так, чтобы получился огонь. Это значительно более естественный для человека способ (как-никак весь реальный мир устроен таким образом). Первым популярным применением этого метода была корзина для удаления файлов в компьютерах *Macintosh* (начиная с *Windows 95*, такая корзина стала стандартом и в *Windows* мире, хотя присутствовала она и раньше). Чтобы не пересказывать уже известное, ограничусь констатацией того простого факта, что если перетащить в неё пиктограмму файла, этот файл будет фактически стерт.

Очевидно, что даже такое простое действие, как стирание файла, на самом деле состоит из многих малых, уже не делимых, действий (жестов). При этом для ускорения мыслительной работы пользователя необходимо не только сокращать количество этих жестов, но и делать эти жесты более простыми.

Помимо того, что этот метод сам по себе состоит из небольшого количества жестов, в определенных ситуациях он оказывается еще короче. Дело в том, что когда расположение корзины (пусть даже и в общих чертах) пользователю известно, процесс удаления файла начинает состоять из одного *единого действия*, т.е. пользователь выбирает файл, высматривает корзину и перетаскивает туда файл одним движением (основной признак единого действия). Более того. Несмотря на то, что пример с корзиной наиболее известен, назвать его оптимальным нельзя. Зачастую задача не так однозначна – пользователь

не только может сделать с объектом что-либо одно, но может сделать несколько разных действий. Например, одно и то же действие (перетаскивание) работает и при удалении, и при перемещении файла. Более того, если перетащить файл в окно электронного письма, которое пользователь в данный момент пишет, файл будет вставлен в письмо как вложение. Это значит, что непосредственное манипулирование позволяет серьезно снизить как количество команд в системе, так и длительность обучения.

Потеря фокуса внимания

Пользователи работают с системой отнюдь не всё время, в течение которого они работают с системой. Это внешне парадоксальное утверждение имеет вполне разумный смысл. Дело в том, что пользователи постоянно отвлекаются.

Таким образом, необходимо максимально облегчать возвращение пользователей к работе и проектировать интерфейс так, чтобы пользователи возможно меньше о нем думали. Понятно, что создание «бездумного» интерфейса задача всеобъемлющая. Мы поговорим о том, как сделать максимально легким возвращение пользователей к работе.

Итак, для продолжения работы пользователь должен знать:

- на каком шаге он остановился;
- какие команды и параметры он уже дал системе;
- что именно он должен сделать на текущем шаге;
- куда было обращено его внимание на момент отвлечения.

Предоставлять пользователю всю эту информацию лучше всего визуально. Разберем это на примере.

Чтобы показать пользователю, на каком шаге он остановился, традиционно используют конструкцию «Страница N из N». К сожалению, эта конструкция работает не слишком эффективно, поскольку не визуальна. Однако существуют и визуальные способы. Например, когда читатель держит в руках книгу, он может понять, в какой её части он находится, по толщине левой и правой части разворота. Можно воспользоваться этой метафорой и на экране: варьировать толщину левых и правых полей окна.

Разумеется, эти методы подходят только для экранных форм. В иных случаях нужно просто делать так, чтобы все стадии процесса выглядели по-разному, благодаря чему хотя бы опытные пользователи, знающие облик всех состояний, могли бы сразу определять текущий шаг.

Показ пользователю ранее отданных им команд чрезвычайно проблематичен. Размеры экрана ограничены, так что почти всегда просто не хватает места для того, чтобы показать всё необходимое. Зачастую единственным выходом из этого положения является максимальное облегчение перехода к предыдущим экранам, да и то это работает только с экранными формами.

Напротив, показывать пользователю, что именно он должен сделать на текущем шаге процедуры, обычно удается легче. С другой стороны, это очень сильно зависит от сущности задачи, так что тут трудно порекомендовать что-либо конкретное.

И, наконец, четвертый пункт: показ пользователю, куда было обращено его внимание на момент отвлечения. Тут есть одна тонкость – обычно фокус внимания совпадает с фокусом ввода. Соответственно, нужно делать фокус ввода максимально более заметным. Легче всего добиться этого цветовым кодированием активного элемента. Есть и другой метод – если количество элементов на экране невелико, пользователь быстро находит активный элемент. Таким образом, просто снизив насыщенность экрана элементами, можно значительно облегчить пользователю возвращение к работе.

Длительность физических действий пользователя, прежде всего, зависит от степени автоматизации работы и степени необходимой точности работы. Об автоматизации что-либо конкретное сказать сложно. Понятно, что чем больше работы делает компьютер, тем лучше. Непонятно только, как это можно универсально описать, поскольку степень автоматизации очень сильно зависит от автоматизируемого процесса. С точностью все гораздо проще.

Быстрый или точный

Любое физическое действие, совершаемое с помощью мускулатуры, может быть *или* точным *или* быстрым. Вместе точность и быстрота встречаются исключительно редко, поскольку для этого нужно выработать существенную степень автоматизма. Объясняется это сугубо физиологическими факторами: при резком движении невозможно быстро остановиться, соответственно, точное движение должно быть плавным и замедленным. Таким образом, чтобы физическое действие пользователя было быстрым, оно не должно быть точным. Пользователь, как правило, управляет компьютером двумя способами, а именно мышью и клавиатурой. Клавиатура не требует особой точности движений – неважно, быстро нажали клавишу или медленно, равно как сильно или слабо. Мышь, напротив, инерционна – есть

разница между медленным её перемещением и быстрым, сильным приложенным усилием и слабым. Именно поэтому оптимизация использования мыши в системе может существенно повысить общую скорость работы.

Мышь не является прецизионным инструментом. Проверить это очень легко – попробуйте мышью нарисовать ровный круг. Соответственно, мышь не предназначена для очень точных, в 1 или 2 пикселя, манипуляций, например, в графических программах всегда есть возможность перемещать объекты клавишами со стрелками. Именно поэтому любой маленький интерфейсный элемент будет всегда вызывать проблемы у пользователей. Более того. Еще в 1954 году Поль Фитс (Paul Fitts) сформулировал правило, в наиболее практичной формулировке ставшее известным как Закон Фитса:

Время достижения цели прямо пропорционально дистанции до цели и обратно пропорционально размеру цели

$$T_{дост\ цели} = a + b \log_2 (D / S + 1)\ мс,$$

Где ***a*** и ***b*** - устанавливаются опытным путем по параметрам производительности человека. Для практического использования можно принять: ***a*** = 50, ***b*** = 150, ***D*** – дистанция от курсора до цели, **S** – размер цели по направлению движения курсора.

Популярно говоря, лучший способ повысить доступность кнопки заключается в том, чтобы делать её большой и располагать ближе к курсору.

У этого правила есть два не сразу заметных следствия. Чтобы «бесконечно» ускорить нажатие кнопки, её, во-первых, можно сделать бесконечного размера и, во-вторых, дистанцию до неё можно сделать нулевой.

Кнопка бесконечного размера. При подведении курсора к краю экрана он останавливается, даже если движение мыши продолжается. Это значит, что кнопка, расположенная впритык к верхнему или нижнему краю экрана, имеет бесконечную высоту (равно как кнопка у левого или правого края имеет бесконечную ширину). Таким образом, скорость достижения такой кнопки зависит только от расстояния до неё и точности выбора начального направления движения. Понятно, что кнопка, расположенная в углу экрана, имеет

бесконечные размеры (не важно даже, с какой точностью перемещали мышь). Для достижения такой кнопки от пользователя требуется всего лишь дёрнуть мышь в нужном направлении, не заботясь о её скорости и не делая попыток остановить её в нужном месте. Это делает такие кнопки наиболее доступными для пользователя, жалко даже, что у экрана всего четыре угла. Именно поэтому, например, меню MacOS многократно эффективней меню Windows: если в MacOS меню всегда расположено впритык к верхнему краю экрана, то в Windows меню отделено от края экрана полосой заголовка окна программы (Title Bar).

Рис. 2. Панель задач (Taskbar) в Windows вызывает удивление – оно расположено впритык к краю экрана, но кнопки отделены от края экрана тремя пустыми пикселями. И скорость работы снижается, и место теряется. © Microsoft.

Нулевая дистанция до кнопки. Рассмотрим контекстное меню, вызываемое по нажатию правой кнопки мыши. Оно всегда открывается под курсором, соответственно расстояние до любого его элемента всегда минимально. Именно поэтому контекстное меню является чуть ли не самым быстрым и эффективным элементом. Но не надо думать, что уменьшать расстояния до цели можно только с контекстными меню. Есть еще диалоговые окна. Они тоже всегда контекстнозависимы, не бывает окон, открывающихся самопроизвольно (на самом деле такие окна есть, но это уже другая история). По умолчанию они открываются в центре экрана, но это легко можно изменить. Открывать их под курсором гораздо лучше, именно потому, что дистанция до их кнопок сокращается, что хорошо не только тем, что перемещать курсор нужно меньше, но также тем, что пользователю сразу становится понятна связь между его действиями и появлением диалогового окна.

Открывайте новые диалоговые окна не в центре экрана, а в центре текущего действия пользователя (если они не будут перекрывать важную информацию на экране, разумеется)

Теперь вернемся к клавиатуре. Как уже было сказано, она не требует особенной точности движений и, как таковая, обеспечивает

большую скорость работы. Тем не менее, и она не без проблем. Во-первых, изначально она не предназначена для перемещения фокуса ввода по экрану, что приводит к существенным трудностям (в том смысле, что самопроизвольно клавиатура не позволяет перемещать фокус с достаточной эффективностью – для этого надо специально проектировать экран). Если клавиатура не работает, приходится пользоваться мышью, но перемещение руки с клавиатуры на мышь и потом обратно занимает почти секунду, что слишком много. Во-вторых, работа с клавиатурой подразумевает использование горячих клавиш (именно потому, что перемещение по экрану с клавиатурой затруднено). Но хотя горячие клавиши существенно увеличивают скорость работы, плохо то, что их трудно запомнить. Таким образом, они являются прерогативой опытных пользователей и для многих людей неприемлемы. Более того, их популярность во многом основывается на субъективных критериях: воспринимаемая пользователем скорость работы с клавиатуры выше, чем скорость работы с мышью (хотя секундомер показывает наоборот). Это значит, что на клавиатуру особо рассчитывать не стоит: помимо набора текста большинство людей пользуются только клавишами пробела и возврата каретки. Тем не менее, игнорировать возможности клавиатуры не следует.

Длительность реакции системы

Часто пользователи надолго прерывают свою работу. Помимо потери фокуса внимания, о котором уже сказано, это плохо тем, что лишенная руководства система начинает простаивать. Разумеется, мы ничего не можем сделать с этой ситуацией: странно было бы, если бы, как только пользователь отходил от компьютера, система, скажем, начинала бы форматировать жесткий диск. Тем не менее, несомненно, и другое: пользователь нередко отвлекается не потому, что появляются внешние раздражители, а потому, что система не реагирует на внешний раздражитель в лице пользователя. Попросту говоря, система делает что-либо длительное. Ни один же человек в здравом уме не будет упорно смотреть в экран, зная, что система будет готова к приему новых команд не ранее, чем через пять минут. Соответственно, человек отвлекается. Проиллюстрировать это очень удобно на процессе печати. Печать документа в сто страниц даже на быстрых принтерах занимает существенное время, соответственно, большинство людей, отправив такой документ в печать, начинают бездельничать, поскольку, чтобы начать следующее действие в их

трудовом процессе, им нужна распечатка, которой ещё нет. Проблема в том, что сразу после того, как человек отвлекается, системе зачастую, во что бы то ни стало, начинает требоваться что-либо от человека. Человек же, уверенный в том, что система работает, уходит в другую комнату. Таким образом, человек и система бездельничают. При этом раздражение человека, вернувшегося с обеденного перерыва и вместо распечатанного документа нашедшего диалоговое окно с вопросом «Вы уверены?», обычно оказывается безмерным. Это делает всегда верным следующее правило: если процесс предположительно будет длительным, система должна убедиться, что она получила всю информацию от пользователя до начала этого процесса.

Есть другое решение этой проблемы: система может считать, что если пользователь не ответил на вопрос, скажем, в течение пяти минут, то его ответ положительный. Таким образом, тот же самый сценарий решается по другому: пользователь отправляет документ на печать и уходит, система спрашивает «Вы уверены?» и ждет пять минут, после истечения этого времени она начинает печать. Этот метод вполне работоспособен, так что им стоит пользовать всегда, когда невозможен первый метод (разумеется, за исключением случаев, когда ответ *Да* может иметь катастрофические последствия).

Убирайте с экрана все диалоги с вопросами, на которые в течение пяти минут не был дан ответ

Есть и другая причина отвлечения пользователя. Пользователь запускает какой-либо процесс. Система показывает ему индикатор степени выполнения. Процент выполнения за минуту едва доходит до четверти размера индикатора. Пользователь экстраполирует эти данные и резонно решает, что у него есть три минуты, чтобы размяться. Однако, как только он отходит от компьютера, процент выполнения с нечеловеческой скоростью начинает расти и за секунду доходит до максимума. Процесс успешно заканчивается, а пользовать еще три минуты бездельничает.

Происходят подобные случаи исключительно потому, что индикаторы степени выполнения обычно рассматриваются программистами не как показатели *процента* выполнения задачи, но как индикаторы того, что система *вообще* работает. Для них это очень удобно: поскольку единый с точки зрения пользователя процесс часто состоит из многих принципиально разных системных процессов, выполняю-

щихся с разной скоростью, можно не утруждаться, стараясь так сбалансировать рост индикатора, чтобы он всё время происходил с одинаковой скоростью. Если этого не сделать процесс принимает довольно комичные формы, так, бывают индикаторы выполнения, которые сначала растут, потом снижаются, потом опять вырастают. Проблема в том, что пользователи рассматривают такие индикаторы именно как способ узнать, когда процесс завершится. Так что врать пользователю тут нехорошо.

Количественный анализ интерфейса

Количественные методы помогают свести спорные вопросы в оценке качества интерфейса к простым вычислениям.

Одним из лучших подходов к количественному анализу моделей интерфейсов является классическая модель *GOMS (the model of goals, objects, methods and selection rules)* - модель целей, объектов, методов и выбор правил.

Эта модель основана на оценке скорости печати.

Время, требуемое для выполнения какой-то задачи системой пользователь-компьютер, является суммой всех временных интервалов, которые потребовались системе на выполнение элементарных жестов, составляющих данную задачу.

Лабораторным путем установлены стандартные средние интервалы для некоторых жестов, выполняемых различными пользователями:

K = 0,2 с – нажатие клавиши;

P = 1,1 с – указание (на какую-то позицию на экране монитора);

H = 0,4 с – перемещение (руки с клавиатуры на ГУВ или с ГУВ на клавиатуру;

M = 1,35 с – ментальная;

R – ответ (время ожидания ответа компьютера).

Расчет по модели *GOMS.*

Основные правила, позволяющие определить, в какие моменты будут проходить ментальные операции, представлены в табл. 1.

Таблица 1

Правило 0 Начальная расстановка операторов M	Оператор M устанавливается перед всеми операторами K (нажатие клавиши) и P, предназначенными для выбора команд, если P указывает на аргументы этих команд оператор M не ставится

Правило 1 Удаление ожидаемых операторов М	Если оператор, следующий за оператором М ожидаемый с точки зрения оператора, предшествующего М, то этот оператор М может быть удален и последовательность РМК превращается в РК
Правило 2 Удаление операторов М внутри когнитивных единиц	Если строка типа МКМКМК… принадлежит когнитивной единице, то следует удалить все операторы М, кроме первого. Когнитивной единицей является непрерывная последовательность вводимых символов, которые могут образовывать название команды или аргумент.
Правило 3 Удаление М перед последовательными разделителями	Если оператор К означает лишний разделитель, стоящий в конце когнитивной единицы (например разделитель команды, следующий сразу за разделителем аргумента этой команды) то следует удалить оператор М, стоящий перед ним.
Правило 4 Удаление операторов М, которые являются прерывателями команд	Если оператор К является разделителем, стоящим после постоянной строки (например, название команды или любая последовательность символов, которая каждый раз вводится в неизменном виде), то следует удалить оператор М, стоящий перед ним. Но если оператор К является разделителем для строки аргументов или любой другой изменяемой строки, то М следует сохранить перед ним.
Правило 5 Удаление перекрывающихся операторов М	Любую часть операторов М, которая перекрывает оператор R, означающий задержку, связанную с ожиданием ответа компьютера, учитывать не следует.

Пример расчета

Исходные данные

Имеется два возможных варианта интерфейса пользователя созданных для задачи перевода температуры из шкалы по Цельсию в шкалу по Фаренгейту и наоборот.

Вариант 1. Диалоговое окно

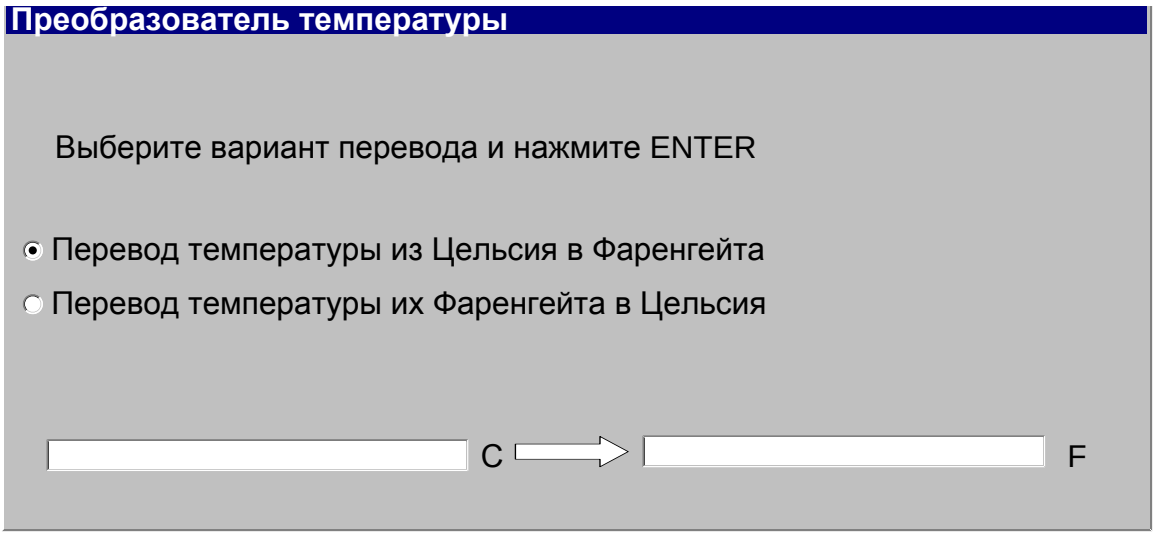

Рис. 2.1 Диалоговое окно

Для перевода температуры в данном варианте необходимо выполнить следующие операции:

- Перемещение руки к ГУВ (Н);
- Перемещение курсора к необходимому переключателю в группе (НР);
- Нажатие на необходимый переключатель (НРК);
- Перемещение рук снова к клавиатуре (НРКН);
- Ввод 4-х символов (НРКНКККК);
- Нажатие клавиши <Enter> (НРКНКККККК);

Применим правило 0 для расстановки индексов М и получим выражение:

НМРМКНМКМКМКМК.

В соответствии с правилом 1 РМК заменим на РК и в соответствии с правилом 2 удалим операторы М внутри когнитивных единиц. Выражение приобретает вид:

НМРКНМКККККМК.

Заменим символы на временные интервалы: К=0,2; Р=1,1; Н=0,4; М=1,35.

Теперь сложим операторы и вычислим сумму времен:

Н+М+Р+К+Н+М+К+К+К+К+М+К =
0,4+1,35+1,1+0,2+0,4+1,35+(4х0,2)+1,35+0,2 = 7,15 с.

Измерение эффективности интерфейса

Если имеется модель интерфейса, то с помощью GOMS можно определить время, необходимое пользователю для выполнения любой, четко сформулированной задачи, для которой данный интерфейс предусмотрен. Однако модели анализа не могут дать ответ на вопрос

о том, насколько быстро должен работать интерфейс. Чтобы ответить на него, необходимо воспользоваться мерой, применяемой в теории информации.

Чтобы сделать правильную оценку времени, необходимого для выполнения задачи с помощью самого быстрого интерфейса, прежде всего, следует определить минимально количество информации, которое пользователь должен ввести, чтобы выполнить задачу. Это минимальное количество не зависит от модели интерфейса. Если методы работы, используемые в предполагаемом интерфейсе, требуют введения такого количества информации, которое превышает минимальное, это означает, что пользователь делает лишнюю работу и поэтому интерфейс следует усовершенствовать. С другой стороны, если от пользователя требуется ввести именно то количество информации, которое необходимо для выполнения задачи, то для этой задачи интерфейс нельзя сделать более производительным путем изменения количества информации. В этом случае пути улучшения интерфейса (а также много путей для ухудшения) все же остаются, но, по крайней мере, данная цель повышения производительности будет уже достигнута.

Информационно-теоретическая производительность определяется так же, как понятие производительности определяется в термодинамике – отношением мощности на выходе к мощности на входе процесса. Если в течение какого-то периода времени электрогенератор, работающий от двигателя мощностью в 1000 ватт, производить 820 ватт, то он имеет производительность 820/1000=0.82. Производительность также часто обозначается через проценты.

Информационная производительность интерфейса E определяется как отношение минимального количества информации, необходимого для выполнения задачи, к количеству информации, которое должен ввести пользователь. Так же как и в отношении физической производительности, параметр E может изменяться в пределах от 0 до 1. Если никакой работы для выполнения задачи не требуется или работа просто не производится, то производительность составляет 1. (Чтобы избежать деления на 0)

Производительность E может равняться и 0 в случаях, когда пользователь должен ввести информацию, которая бесполезна для выполнения задачи.

В параметре E учитывается только информация, необходимая для задачи и информация, вводимая пользователем.

Так для выбора одного варианта из четырех возможных достаточно двух бит информации, из восьми – трех, из 16 – 4-х и т.д.

Суммарное количество передаваемой информации определяется, как:

$\log_2 n$

Количество информации для каждого варианта определяется как:

$(1/n) \log_2 n$

(1)

Если вероятности для каждого варианта или элемента не являются равными и

i- й вариант имеет вероятность P(i), то информация, передаваемая этим вариантом определяется, как:

$p(i) \log_2 (1/p(i)$

(2)

Количество информации по всем вариантам является их суммой по всем вариантам выражения (2), которое при равновероятных вариантах сводится к выражению (1).

Для извлечения логарифма по основанию 2 можно воспользоваться выражением

$\log_2 (x) = \ln(x) / \ln(2)$

2. Человеческие ошибки

Важным критерием эффективности интерфейса является количество человеческих ошибок. В некоторых случаях одна или две человеческих ошибки погоды не делают, но только тогда, когда эти ошибки легко исправляются. Однако часто минимальная ошибка приводит к совершенно катастрофическим последствиям, например, за одну секунду оператор в банке может сделать кого-то богаче, а банк, в свою очередь, беднее (впрочем, обычно беднее становятся все). Классический сюжет из жизни летчика, который после взлета хотел убрать шасси, но вместо этого включил систему аварийного катапультирования, возник отнюдь не на пустом месте.

Природа существования ошибок.

Вначале необходимо сказать главное: человек при работе с компьютером постоянно совершает ошибки, сами мы, например, делали их не раз и не два!

Дело в том, что компьютеры (как и все сложные технические системы) вообще не могут быть используемы человеком без совер-

шения ошибок. Компьютеры требуют от человека точности, логического мышления, способности абстрагироваться от идей реального мира. Человек же практически на это не способен. Человек не цифровая система, неспособная на ошибку, но система аналоговая. Именно благодаря этому он плох в логике, зато имеет интуицию, не приспособлен к точности, зато может подстраиваться к ситуации, слабо абстрагируется, зато хорошо разбирается в реальном мире.

Попробуйте вслушаться в любой разговор между людьми. Вы обнаружите, что он полон запинок, пауз, фраз оборванных на середине или даже полностью отмененных последующими словами. С точки зрения компьютера такой разговор подобен смерти, для нас же это естественное положение вещей. Суммируя, можно сказать, совершение ошибок есть естественное занятие человека. А раз ошибки естественны, значит система, неспособная сама их обнаружить и исправить, порочна. Таким образом, человеческих ошибок не бывает. Бывают ошибки в проектировании систем. Сам термин «человеческая ошибка» до сих пор существует только по двум причинам. Во-первых, люди в ошибках системы склонны винить себя, поскольку по собственному эгоцентризму полагают, что подобные вещи происходят только с ними. Во-вторых, существующее положение вещей очень выгодно всякому руководству: гораздо легче уволить кого-либо, нежели признать, что система спроектирована плохо.

Под словосочетанием «человеческая ошибка» нужно понимать *«действие пользователя, не совпадающее с целью действий этого пользователя»*.

Типы ошибок

Классификаций человеческих ошибок существует великое множество, чуть ли не каждый автор, пишущий на эту тему, создает новую классификацию. Наибольшее количество человеческих ошибок при пользовании ПО раскладывается на четыре типа (сильно упрощенно, разумеется):

Ошибки, вызванные недостаточным знанием предметной области. Теоретически, эти ошибки методологических проблем не вызывают, сравнительно легко исправляясь обучением пользователей. Практически же, роль этих ошибок чрезвычайно велика – никого не удивляет, когда оператора радарной установки перед началом работы оператором долго учат работать, и в то же время все ожидают должного уровня подготовки от пользователей ПО, которых никто никогда ничему целенаправленно не обучал. Еще хуже ситуация с

сайтами, у которых даже справочной системы почти никогда не бывает.

Опечатки происходят в двух случаях:

- когда не все внимание уделяется выполнению текущего действия (этот тип ошибок характерен, прежде всего, для опытных пользователей, не проверяющих каждый свой шаг);
- когда в мысленный план выполняемого действия вклинивается фрагмент плана из другого действия (происходит преимущественно в случаях, когда пользователь имеет обдуманное текущее действие и уже обдумывает следующее действие).

Ошибки, вызванные не считыванием показаний системы, которые одинаково охотно производят как опытные, так и неопытные пользователи. Первые не считывают показаний системы потому, что у них уже сложилось мнение о текущем состоянии, и они считают излишним его проверять, вторые – потому что они либо забывают считывать показания, либо не знают, что это нужно делать и как.

Моторные ошибки, количество, которых фактически пренебрежимо мало, но к сожалению, не так мало, чтобы вовсе их не учитывать. Сущностью этих ошибок являются ситуации, когда пользователь знает, что он должен сделать, знает, как этого добиться, но не может выполнить действие нормально из-за того, что физические действия, которые нужно выполнить, выполнить трудно. Так, никто не может с первого раза (и со второго тоже) нажать на экранную кнопку размером 1 на 1 пиксель. При увеличении размеров кнопки вероятность ошибки снижается, но почти никогда не достигает нуля. Соответственно, единственным средством избежать этих ошибок является снижение требований к точности движений пользователя.

В целом, в отношении человеческих ошибок, ситуация с ПО и сайтами лучше ситуации с управлением динамическими процессами (самолеты, производственные линии, энергетические станции и т.д.), которая и составляла основное приложение усилий инженерной психологии. С одной стороны, фактически отсутствует обучение пользователей перед работой, зато с другой – нет ни постоянно изменяющейся внешней среды, ни лимита времени.

Тут уместно упомянуть одно из важных понятий инженерной психологии, а именно бдительность, т.е. способность оператора в течение продолжительного времени направлять существенную часть своего внимания на состояние системы. Как показывает практика, ни

один человек не способен долгое время обеспечивать бдительность без существенных потерь: мозг стремится найти себе более интересное занятие, отчего накапливается усталость и стресс. Нечего и говорить, что эти «усталость, раздражение и стресс вообще» никаким образом не приводят к получению удовольствия при работе с системой. Но как только бдительность снижается, количество ошибок возрастает в разы. Таким образом, проблема состоит в том, что для успешного пользования любой системой необходима определенная степень бдительности, но эта же бдительность пользователям неприятна. В условиях невозможности отбора пользователей (есть люди, способные быть бдительными дольше других), эта проблема не решается вообще. Потенциально, *в систему можно ввести индикатор опасности текущего состояния, при этом пользователь получает право быть не слишком бдительным большую часть времени, но зато получает и обязанность быть максимально собранным, когда горит «красная лампочка»*. С некоторыми оговорками, этот метод может быть использован (и используется) на атомной станции, но совершенно непонятно, как его можно применить, например, в электронной таблице, в которой всего два действия пользователя способны испортить целый документ.

Важно также понимать, что крайне ценная возможность последующей отмены (Undo) деструктивных действий сама по себе не служит уменьшению количества человеческих ошибок, но помогает только уменьшить урон от них. В действительности надо стремиться минимизировать количество ошибок, поскольку только это позволяет сберечь время (т.е. повысить производительность) и сделать пользователей более счастливыми за счет отсутствия дискомфорта.

Суммируя, при борьбе с ошибками нужно направлять усилия на:

- *плавное обучение пользователей в процессе работы*
- *снижение требований к бдительности*
- *повышение разборчивости и заметности индикаторов.*
- *снижение чувствительности системы к ошибкам.*

Для этого есть три основных способа, а именно:

- *блокировка потенциально опасных действий пользователя до получения подтверждения правильности действия*
- *проверка системой всех действий пользователя перед их принятием*

- *самостоятельный выбор системой необходимых команд или параметров, при этом от пользователя требуется только проверка.*

Самым эффективным является третий способ. К сожалению, этот способ наиболее труден в реализации. Разберем эти три способа подробнее.

Блокировка потенциально опасных действий до получения подтверждения

Команда удаления файла в любой операционной системе снабжена требованием подтвердить удаление. Эта блокировка приносит пользу только начинающим пользователям, которые проверяют каждый свой шаг.

Для опытных пользователей это диалоговое окно с требованием подтверждения не работает. Во-первых, оно не защищает нужные файлы. Во-вторых, оно без пользы отвлекает пользователя и тратит его время.

В то же время некоторую пользу от этого метода получить можно. Для этого только *надо требовать подтверждения не после команды пользователя, а до неё*. Предположим, чтобы удалить файл, нужно сначала в контекстном меню выбрать команду Разблокировать, после чего выбрать этот же файл и запустить процесс его удаления (неважно, с клавиатуры или из меню). В этом случае от пользователя действительно требуется подтвердить удаление, поскольку эти два действия напрямую не связаны друг с другом – если в одном из них была допущена ошибка, файл удалить не удастся.

К сожалению, этот принцип применять довольно тяжело. Дело в том, что ситуации, подобные описанной, встречаются довольно редко. *Гораздо чаще приходится защищать не отдельные объекты (файлы, окна и т.п.), но отдельные фрагменты данных (например, текст и числа в полях ввода)*. Проблема состоит в том, что понятного и удобного элемента управления для этой цели нет. *Единственным выходом служит скрытие потенциально опасных данных от пользователя до тех пор, пока он сам не скомандует системе их показать*. Выход же этот отнюдь не идеальный, поскольку некоторым пользователям никогда не удастся понять, что, помимо видимых значений, есть еще и невидимые данные.

Не делайте опасные для пользователя кнопки кнопками по умолчанию

Также к этому типу блокировки относится снятие фокуса ввода с кнопок конечных действий, чтобы пользователь не мог, не разобравшись, нажать на кнопку *Enter* и тем самым начать потенциально опасное действие. Действительно, если пользователям приходится прилагать какие-либо усилия, чтобы запустить действие, есть надежда, что во время совершения этих усилий он заметит вкравшуюся ошибку. Обычно проще всего в опасных случаях не делать главную кнопку кнопкой по умолчанию. Также, важно не делать кнопку Отмена кнопкой по умолчанию (как часто случается). Если это сделать, пользователи будут ошибочно закрывать окно, т.е. одна ошибка заменит другую.

Проверка действий пользователя перед их принятием

Этот метод гораздо лучше блокировки, но он тоже не без недостатка: трудно проверять команды. Наиболее популярны два универсальных и работающих способа проверки. Во-первых, это меню. *В случаях, когда пользователь выбирает команду из списка, система может без труда делать так, чтобы в этот список попадали только корректные команды* (это вообще достоинство любого меню). Во-вторых, *если действие запускается непосредственным манипулированием объектами, можно индицировать возможные действия изменением поведения этих объектов*. Например, *если бы форматирование диска запускалось не нажатием кнопки, а перенесением пиктограммы диска в область форматирования, можно было бы показывать пользователю, как с выбранного диска исчезают все файлы и папки.* При этом не только снизилась бы вероятность ошибочного форматирования диска, поскольку перенести объект в другую область труднее, чем просто нажать на кнопку, но при этом исчезла бы необходимость предупреждать пользователя о грядущей потере данных с помощью дурацкого сообщения. Проверкой всех действий пользователя перед их принятием *можно также успешно защищать вводимые пользователем данные, в особенности данные численные*. Дело в том, что большинство численных данных имеют некий диапазон возможных значений, так что даже в ситуациях, когда невозможно проверить корректность данных, можно, по крайней мере, *убедиться, что они попадают в нужный диапазон*. В большинстве ОС есть специальный элемент управления, именуемый *крутилкой.* Фактически это обычное поле ввода, снабженное двумя кнопками для модификации его содержимого (в сторону уменьшения и увеличения). Интере-

сен он тем, что пользователь может не пользоваться клавиатурой для ввода нужного значения, взамен клавиатуры установив нужное значение мышью. Этот элемент имеет то существенное достоинство, что при использовании мыши значение в этом элементе всегда находится в нужном диапазоне и обладает нужным форматом.

Всегда показывайте границы диапазона во всплывающей подсказке

Но что делать, *если пользователь ввёл некорректное число с клавиатуры*? Ответ прост. Для этого *надо индицировать возможную ошибку изменением начертания шрифта на полужирное* в обычных программах (иное проблематично), *а в случае сайта – заменой цвета фона этого элемента на розовый* (благо это нетрудно сделать через таблицу стилей).

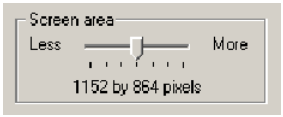

Рис. 3. Ползунок.

В тех же случаях, когда количество возможных значений невелико, лучше использовать другой элемент управления – ползунок. Мало того, что он позволяет устанавливать только определенные значения (с этим справился бы и выпадающий список или комплект переключателей), но *он позволяет пользователю видеть взаимосвязь возможных значений* и при этом использование этого элемента понятно даже новичку.

Самостоятельный выбор команд

И, наконец, самый эффективный способ. Системе, как никак, лучше знать, какие именно команды или параметры для неё пригодны. Соответственно, чем меньше действий требуется совершить пользователю, тем меньше вероятность ошибки (при этом пользователь, которого избавили от рутинной работы, уже радуется). Вопрос состоит в том, как системе узнать, что именно нужно пользователю.

Проиллюстрировать сферу применения данного метода удобно на примере печати. MS Windows User Experience заставляет использовать только один способ что-либо напечатать. Существует две команды меню Файл, а именно Печать и Параметры печати. Обе команды вызывают одноименные диалоговые окна. Проблема заключается в том, что обилие элементов управления замедляет восприятие этих окон и увеличивает вероятность ошибки.

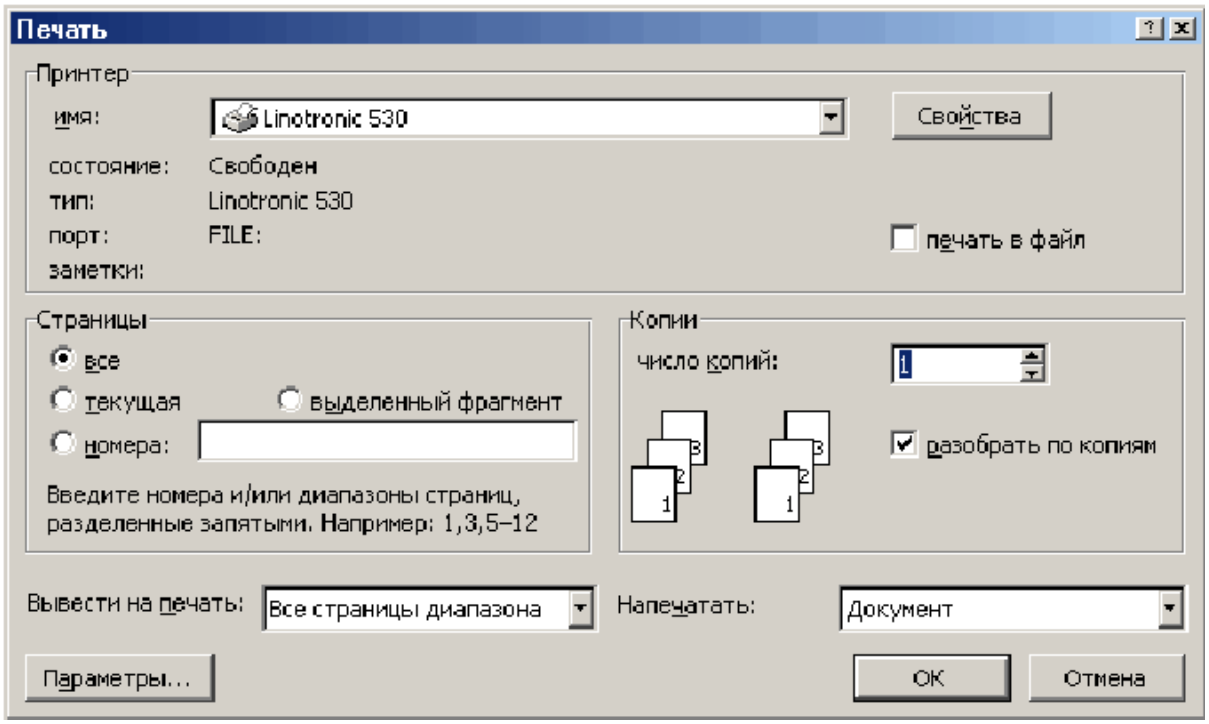

Рис. 4. Диалоговое окно печати в MS Word. © Microsoft.

Разберём это подробнее. Итак, чем меньше элементов управления, тем меньше вероятность ошибки. Система может уменьшить число элементов, если она знает сама, какими именно параметрами она должна руководствоваться. Главной причиной появления этих диалоговых окон является печать нескольких копий. Причем есть простая зависимость – «количество копий обратно пропорционально частоте печати такого количества», т.е. сто копий печатают примерно в сто раз реже, чем печатают одну копию. Стандартное диалоговое окно печати содержит также область выбора принтера из числа установленных в системе. Большинство же пользователей имеет только один принтер. Так зачем заставлять это большинство каждый раз вдумчиво воспринимать совершенно не нужные им элементы интерфейса?

Интересно, что всё это прекрасно понимают в Microsoft. В каждой включенной в комплект MS Office программе на панели инструментов есть кнопка, нажатие на которую вызывает печать одного экземпляра с текущими настройками. Это, впрочем, тоже нехорошо. Во-первых, кнопка называется Печать, каковое название конфликтует с такой же командой в меню (называть кнопку Печать одного экземпляра с текущими настройками неприлично). Сама же по себе идея иметь в программе две кнопки с одинаковыми названиями и разным действием порочна. Во-вторых, нормальная программа должна иметь

меню, содержащее полную функциональность, и панель инструментов, представляющую собой выжимку из меню. А здесь получается, что в панели инструментов есть команда, которую нельзя вызвать никаким иным способом.

А теперь представьте себе другой вариант. *В меню есть те же две команды – Печать и Параметры печати. Выбор первой команды вызывает немедленную печать документа с текущими настройками. Выбор второй вызывает диалог со всеми доступными параметрами печати,* т.е. в системе с одним принтером никогда не появится ненужная группа элементов. Если же пользователь начинает проявлять буйную активность и печатать несколько копий разом, включается другой механизм. В первый раз, когда пользователь меняет число копий в окне настроек печати, программа запоминает его действие и при следующем выборе команды Печать выводит диалоговое окно со всего двумя элементами управления – полем ввода, в котором уже стоит число копий (которое было запомнено в предыдущий раз) и кнопкой ОК. Поскольку программа не может быть уверена в правильности числа копий, цифру лучше всего выводить нестандартным цветом, чтобы привлечь внимание пользователя. И так до тех пор, пока пользователь два раза подряд не введет единицу в поле ввода (что переводит его в разряд представителей большинства) или не введет новое число копий (каковое и будет запомнено). Причём такая метода применяется абсолютно ко всем возможным настройкам, а не только к числу копий. Таким образом, большинство пользователей становится счастливо, а количество ошибок сокращается, что хорошо.

Суммируя, можно сказать, что система сама может узнать большинство из тех сведений, которые она запрашивает у пользователя. Главными источниками этих сведений являются:

- *здравый смысл разработчика системы*
- *предыдущие установленные параметры*
- *наиболее часто устанавливаемые параметры.*

С другой стороны, применяя этот метод, надо всегда помнить о том, что цель этого метода состоит не в том, чтобы провести пользователя за ручку по программе, оберегая его от всего, что может его испугать, но в том, чтобы сделать пользователя более счастливым. Счастье же достигается вовсе не в покое, но в борьбе с невзгодами. Многим людям будет комфортабельней работать со сложной систе-

мой, нежели со слишком упрощенной. Единственная проблема этого метода заключается в том, что для его использования к проектированию системы нужно подходить значительно более творчески и тщательно, нежели обычно практикуется.

Два уровня ошибок и обратная связь

Помимо классификации человеческих ошибок, приведенной в начале главы, существует ещё одна классификация. В этой классификации ошибки расставлены по уровням их негативного эффекта:

1. Ошибки, исправляемые во время совершения действия, например пользователь перетаскивает файл в корзину и во время перетаскивания замечает, что он пытается стереть не тот файл.	3. Ошибки, которые исправить можно, но с трудом, например реальное стирание файла, при котором никаких его копий не остается.
2. Ошибки, исправляемые после выполнения действия, например, после ошибочного уничтожения файла его копия переносится из корзины.	4. Ошибки, которые на практике невозможно исправить, т.е. ошибки, которые невозможно обнаружить формальной проверкой (т.е. невозможно обнаружить их не случайно). Пример: смысловая ошибка в тексте, удовлетворяющая правилам языка.

Каждый хороший программист, умеющий мыслить системно, знает, что ошибок из четвертого пункта нужно всеми силами избегать, не считаясь с потерями, поскольку каждая такая ошибка обходится гораздо дороже, чем любая ошибка из пункта третьего. Но не каждый хороший дизайнер интерфейса, умеющий мыслить системно, знает, что ошибок из второго пункта нужно всеми силами избегать, поскольку каждая такая ошибка обходится гораздо дороже, чем любая ошибка из первого пункта. Объясняется это просто: дизайн интерфейса гораздо моложе программирования.

С ошибками из четвертого пункта всё ясно. Всякий раз, когда мы теряем возможность, по крайней мере, проверить корректность данных или самой системы, мы вступаем на слишком уж скользкий путь. Межпланетные зонды, из-за ошибок в ПО улетают не туда куда надо, коммерческие договоры, в которых обнаруживаются ошибки

приносят много неприятностей, ошибочные номера телефонов в записной книжке не дают возможности найти абонента – всё это примеры неисправляемых ошибок. Разумеется, такие ошибки всегда обнаруживаются, проблема в том, что к моменту их обнаружения становится поздно их исправлять. Именно поэтому такие ошибки гораздо хуже ошибок, которые исправить трудно, но которые, по крайней мере, сразу видны. Единственной индустрией, научившейся получать пользу от необнаруженных ошибок, является производство почтовых марок – марки с опечатками стоят у филателистов многократно дороже марок без них. Это было знание программистов. Теперь пора перейти к интерфейсу и определить, почему ошибки первого типа («исправляемые во время») гораздо лучше ошибок второго типа («исправляемых после»).

Вообще говоря, объяснение этого факта двояко. Объяснение есть как субъективное, так и объективное, сказать, какое сильнее, затруднительно. При этом объяснения еще и складываются.

Объективное объяснение просто: ошибки, исправляемые после, снижают производительность работы. Как мы уже знаем из предыдущей главы, любое действие пользователя состоит из семи шагов. Всякий раз, когда пользователь обнаруживает, что он совершает ошибку, ему приходится возвращаться назад на несколько этапов. Более того, чтобы исправить совершенную ошибку, от пользователя требуется:

- *понять, что ошибка совершена*
- *понять, как её исправить*
- *потратить время на исправление ошибки.*

В результате значительный процент времени уходит не на действие (т.е. на продуктивную работу), а на исправление ошибок.

Субъективное объяснение ещё проще: ошибки, исправляемые после, воспринимаются пользователем как ошибки. Ошибки же, исправляемые во время, как ошибки не воспринимаются, просто потому, что для пользователей это не ошибки вообще: все человеческие действия до конца не алгоритмизированы, они формируются внешней средой (так не получилось и так не получилось, а вот так получилось). Ошибка же, не воспринимаемая как таковая, пользователей не раздражает, что весьма положительно действует на их субъективное удовлетворение от системы.

Наличие человеческих ошибок, которых нельзя обнаружить и исправить до окончательного совершения действия, всегда свидетельствует о недостаточно хорошем дизайне

Теперь пора сказать, как избавится от ошибок, исправляемых после. Понятно, что исправить что-либо «во время» можно только тогда, когда во время совершения действия видно, что происходит и как это действие повлияет на изменяемый объект. *Соответственно, чтобы дать пользователям исправлять их действия на ходу, этим пользователям надо дать обратную связь*.

К сожалению, это простое соображение имеет существенный недостаток: вводить в систему обратную связь получается не всегда. Дело в том, что её ненавидят программисты. Мотивируют они своё отношение тем, что она плохо влияет на производительность системы. Обычно они обманывают. На самом деле им просто лень её реализовывать. Иногда, впрочем, соображения о производительности системы и вправду имеют место. Так что если вы чувствуете, что программисты правы, когда говорят о том, что система «будет тормозить», вспомните, что производительность связки «система-пользователь» всегда важнее производительности системы просто. Если же и это не помогает, попробуйте спроектировать обратную связь иначе, более скромно. Иногда так получается даже лучше. Например, с помощью ползунков на линейке в MS Word можно менять абзацные отступы, при этом обратная связь есть, но не полная: вместо перманентного переформатирования документа по экрану двигается полоска, показывающая, куда передвинется текст. Благодаря этому изображение на экране особенно не перерисовывается, что хорошо, поскольку такое «дрыганье» раздражает.

3. Обучение работе с системой

В традиционной науке о человеко-машинном взаимодействии роль обучения операторов чрезвычайно велика. Мало того, что в дополнении к самой системе разрабатывается методология обучения её будущих пользователей, так еще и разрабатываются нормативы на пользователей, и если человек будет сочтен неподходящим, к системе его просто не допустят. Напротив, с ПО и сайтами ситуация принципиально иная: как цель ставится возможность работы с системой для любого человека, независимо от его свойств и навыков, при этом целенаправленное обучение пользователей, как правило, не произво-

дится. Всё это делает проблему обучения пользователей работе с компьютерной системой чрезвычайно важной.

Почему пользователи учатся

Есть непреложный закон природы: люди делают что-либо только при наличии стимула, при этом тяжесть действия пропорциональна силе стимул. Применительно к компьютерным системам этот закон действует без каких-либо исключений. Обучение есть действие: если обучаться легко, пользователям будет достаточно слабого стимула, если тяжело, стимул придется увеличивать. Но если стимул для пилота самолета получают преимущественно командными методами (если он не научится определенному минимуму, его просто не допустят до работы), то в случаях компьютерных систем стимул есть вещь почти исключительно добровольная. Это значит, что пользователь обучится пользоваться программой или сайтом только в том случае, если он будет уверен, что это, к примеру, сделает его жизнь легче и приятней.

Пользователь будет учиться какой-либо функции, только если он знает о её существовании, поскольку, не обладая этим знанием, он не способен узнать, что за её использование жизнь даст ему награду. Т.е. одного стимула недостаточно, если пользователь не знает, за что этот стимул дается.

Рассчитывайте на средних пользователей, а не новичков или на профессионалов: средних пользователей, как-никак, абсолютное большинство

Средства обучения

Обычно считается, что в случае ПО есть два способа повысить эффективность обучения (помимо метода «обучения плаванию посредством выбрасывания из лодки»), а именно бумажная документация и «оперативная справка». Но существуют более эффективные способы. Рассмотрим некоторые их них:

- общая «понятность» системы
- обучающие материалы.

Понятность системы

Термин «понятность» включает в себя три составляющих, а именно ментальную модель, метафору, аффорданс и стандарт

Ментальная модель. Зачастую, или, точнее, почти всегда, чтобы успешно пользоваться какой-либо системой, человеку необходимо однозначно понимать, как система работает. При этом необязательно точно понимать сущность происходящих в системе процессов, более того, необязательно правильно их понимать. Это понимание сущности системы называется ментальной моделью.

Метафора. Чтобы научиться пользоваться системой, пользователю нужно построить ментальную модель этой системы. Избавить его и от этой работы можно применением метафоры, которая позволяет пользователю не создавать новую модель, а воспользоваться готовой моделью, которую он ранее построил по другому поводу.

Анализируя опыт применения метафор, можно вывести следующие правила:

- *опасно полностью копировать метафору, достаточно взять из неё самое лучшее*
- *не обязательно брать метафору из реального мира, её смело можно придумать самому*
- *эффективнее всего метафорически объяснять значение отдельных объектов: например, для графической программы слои можно представлять как положенные друг на друга листы стекла (этот пример подходит и для предыдущего пункта)*
- *если метафора хоть как-то ограничивает систему, от неё необходимо немедленно отказаться.*

Аффорданс. В современном значении этого термина аффорданс называется ситуация, при котором объект показывает субъекту способ своего использования своими неотъемлемыми свойствами. Например, надпись «На себя» на двери не является аффордансом, а *облик* двери, который подсказывает человеку, что она открывается на себя, несет в себе аффорданс.

Польза аффорданса заключается в том, что он позволяет пользователям обходиться без какого-либо предварительного обучения, благодаря этому аффорданс является самым эффективным и надежным средством обеспечения понятности.

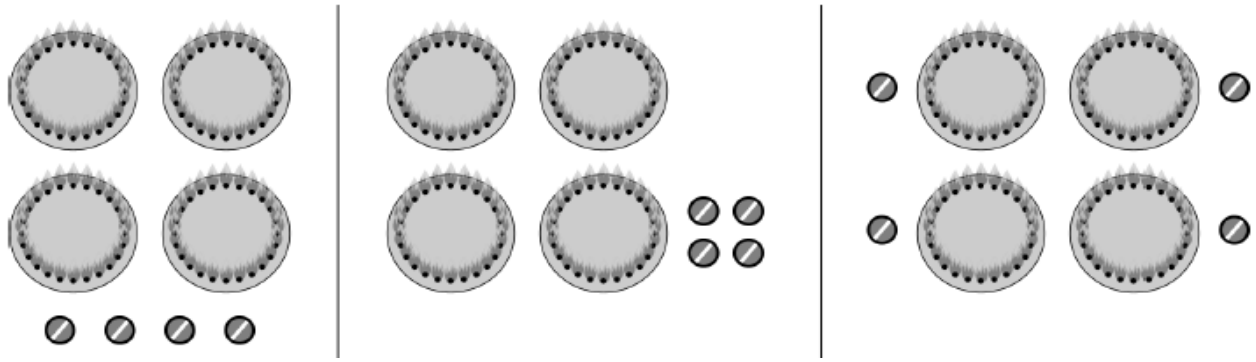

Рис. 6. Отсутствие аффорданса в дизайне кухонной плиты. Слева стандартный вариант, в центре и справа варианты с аффордансом

Способы передачи аффорданса:

- *маппинг, или повторение конфигурации объектов конфигурацией элементов управления (этот способ работает хорошо в реальном мире, но не очень хорошо на экране, поскольку предпочтительней непосредственное манипулирование)*
- *видимая принадлежность управляющих элементов объекту*
- *визуальное совпадение аффордансов экранных объектов с такими же аффордансами объектов реального мира (кнопка в реальном мире предлагает пользователю нажать на неё, псевдотрехмерная кнопка предлагает нажать на неё по аналогии)*
- *изменение свойств объекта при подведении к нему курсора (бледный аналог тактильного исследования).*

Стандарт. Наконец, остался последний, самый мощный, но зато и самый ненадежный способ обучения, а именно стандарт. Дело в том, что если что-либо нельзя сделать «самопроизвольно» понятным, всегда можно сделать это везде одинаково, чтобы пользователи обучались только один раз. Например, кран с горячей водой всегда маркируют красным цветом, а кран с холодной – синим. Частично это соответствует свойствам человеческого восприятия (недаром красный цвет мы называем тёплым, а синий – холодным), но в основном здесь работает привычка.

Как я уже сказал, стандарт штука мощная, но зато ненадежная. Он очень хорошо работает, когда популярен, в противном случае не работает вовсе. Так, с одной стороны, при появлении стандартизированных графических интерфейсов количество программ на душу пользователя увеличилось на порядок (т.е. пользователи получили возможность с тем же уровнем усилий изучать больше программ), с

другой стороны, множество хороших вещей так и не было реализовано, поскольку в нужный момент подходящих стандартов не было. Таким образом, чтобы стандарт заработал, он должен быть популярен. Популярность стандарта может быть достигнута двумя способами: во-первых, он может быть во всех системах, во-вторых, он может быть популярен внутри отдельной системы. Например, стандарт интерфейса MS Windows популярен почти во всех программах для Windows, именно поэтому его нужно придерживаться. С другой стороны, этот стандарт оставляет неопределенным очень многое (никто да не обнимет необъятного), и это многое в разных системах трактуется по-разному. Бесполезно пытаться найти общий знаменатель во всех системах, гораздо эффективнее установить собственный стандарт, не противоречащий стандарту ОС, но дополняющий его; после чего этого стандарта надо придерживаться.

Последовательность в реализации интерфейса есть первое условие качества результата

Конечно, разработка собственного стандарта дело довольно тяжелое. С другой стороны, сказать, что она совсем уж невозможна, тоже нельзя. Во-первых, самое главное уже стандартизовано. Во-вторых, стандарт может развиваться параллельно процессу разработки, при этом поддержание стандарта не стоит почти ничего. Обширный же стандарт вполне окупает вложенные в него усилия ускорением обучения пользователей, не говоря уже о снижении стоимости разработки.

Стандарты, обеспечивающие интерфейсы пользователей с операционной средой

Задачи взаимодействия пользователей с операционной средой. Мобильность информационных систем включает унификацию и сохранение технологии взаимодействия и всех деталей интерфейса пользователей с приложениями при изменении аппаратных и операционных платформ — ***мобильность пользователей.*** Тем самым пользователь не должен переучиваться при переносе определенных прикладных программ и данных на иные платформы и сохраняет опыт и технологию взаимодействия с ними. В результате сокращаются затраты на освоение новых платформ и предотвращаются многие ошибки при использовании приложений [14,15,8].

Парк терминалов для пользователей информационных систем не-

прерывно совершенствуется по своим функциональным характеристикам и аппаратным реализациям. Для сохранения применяемых операционных систем, прикладных программ и баз данных и обеспечения возможности их переноса на иные платформы необходимо выделение и унификация интерфейсных компонентов, организующих их взаимодействие с различными аппаратно-программными реализациями терминалов. Эти интерфейсные компоненты должны согласовывать терминальные методы доступа к функциональным приложениям разнородных средств распределенных информационных систем, отличающихся между собой и использующих различные принципы отображения информации. Для этого необходима унификация концепции, архитектуры, функций и методов визуализации пользовательского интерфейса. ***Задача обеспечения мобильности пользовательского интерфейса включает стандартизацию:***

— визуализации и непосредственного взаимодействия пользователей с различными типами терминалов;

— интерфейсов прикладных программных средств, обеспечивающих визуализацию, с операционной системой;

— интерфейсов программных средств визуализации с приложениями;

— интерфейсов прикладных программных средств и баз данных с операционной системой (API).

Программные средства визуализации и взаимодействия с пользователями по отношению к операционной системе выступают как прикладные программы и их интерфейсы с ней должны соответствовать стандартам API POSIX и в частности стандарту IEEE 1003.4 для систем реального времени. Стандартизация визуализации и непосредственного взаимодействия пользователей с различными типами терминалов затруднена широким спектром функционального назначения и особенностей отображаемой информации и классов информационных систем. Однако при переносе на иные платформы программ и данных определенного назначения их визуализация должна сохраняться на уровне стандартов де-факто. Наибольшие успехи достигнуты в международной стандартизации архитектуры интерфейсов программных средств визуализации с операционными системами. Применение этих стандартов значительно облегчает переносимость пользовательского интерфейса и приложений на иные платформы. Основная совокупность стандартов поддерживает графические пользовательские интерфейсы (Graphical User Interfaces - GUI). По своему функциональ-

ному назначению GUI является альтернативой вводу данных через командную строку. Ключевой проблемой остается выработка единой методологии создания программ графических систем [15,16,17,18].

Графический пользовательский интерфейс делает компьютер более легким и удобным для использования. Для этого он должен быть "дружественным" по отношению к пользователю, предусматривать возможные нужды и функции определенного класса пользователей, предостерегать их от ошибок при работе и защищать от разрушения системы. Он позволяет заменить командное управление с учетом сложного синтаксиса операционной системы, на манипулирование окнами и элементами изображения (иконами) пользовательских интерфейсов. Интерфейс с пользователем должен быть логичным и согласованным, не создавать неожиданные или тупиковые ситуации при любых действиях пользователей в соответствии с инструкциями, иметь соответствующие учебники и руководства для помощи при работе и для предотвращения ошибок. При переносе GUI на иные платформы должна полностью сохраняться визуализация на экране дисплея и функциональные возможности взаимодействия пользователя с графической системой.

Основой GUI являются наборы графических элементов и допустимых операций над ними, меню и области окон для прокрутки изображений. Прикладной программный интерфейс (API) реализует программно-языковые функции для взаимодействия разработчиков приложений с графическими интерфейсами. Для этого программист выбирает и компонует наборы элементов и функций (окон, меню, операций над изображениями, икон), необходимых для создания конкретных информационных систем. Для поддержки экономного и эффективного переноса программных средств, обеспечивающих отображение различных графических образов, применяются стандарты де-факто и де-юре. Стандарты де-факто созданы несколькими рабочими группами фирм и IEEE и имеют целью унифицировать непосредственное формирование изображения на экране дисплея, управление окнами и графическими образами.

Проектирование интерфейсов пользователей состоит в разработке интегрированного набора средств, помогающих разработчику в создании и управлении различными интерфейсами пользователей. Основу пользовательского интерфейса составляют наборы графических элементов и действий над ними, представляемые как меню и системы окон для манипулирования с изображениями. При этом ставится за-

дача отделить процесс создания интерфейса пользователей от разработки прикладных программ, которые не должны связываться с конечными пользователями напрямую. ***Основные особенности современного интерфейса с пользователями*** состоят в следующем [19,20,21]:

— наличие механизмов управления окнами;

— использование готовых графических символов (икон) для отображения управляемых объектов;

— непосредственное манипулирование графическими объектами и окнами посредством "мыши";

— объектно- и проблемно-ориентированное проектирование диалоговых систем.

Для реализации интерфейсов создаются и используются библиотеки технологических интерактивных программ, позволяющих использовать устройства ввода команд управления и графических элементов при наличии обратной связи, отображающей на дисплее результаты манипулирования ими. Для этого разрабатываются модели, методы и языки проектирования интерфейсов пользователей, которые должны соответствовать проблемной области информационной системы. Их можно отнести к компонентам языков четвертого поколения. Разработка систем построения и управления интерфейсами пользователей ведется многими фирмами послойно с частичным учетом стандартов открытых систем и постепенной выработкой ряда стандартов де-факто.

Модели графического пользовательского интерфейса. Разработан ряд моделей графического пользовательского интерфейса, из которых ниже рассматриваются три. **Первая —*концептуальная модель*** может быть детализирована введением пяти уровней взаимодействия пользователей с информационными системами: физического; концептуального; лингвистического; визуального; функционального.

Физический уровень определяет состав технических средств и общетехнические требования к ним, например, расположение клавиш на клавиатуре, характеристики устройств автоматического ввода информации, характеристики мониторов. Физический уровень взаимодействия соответствует нижнему уровню интерфейса пользователя в концептуальной модели ИС.

Концептуальный уровень определяет способ отображения состояния среды, с которой взаимодействует пользователь. Он базируется

на некотором представлении объектов среды через объекты интерфейса, связываемые между собой в программной среде. ***Лингвистический уровень*** определяет все, что связано с обработкой текстовой информации: сообщения программной среды, ввод текстовых команд пользователя, редактирование текста и т.д. Возможности интерфейса в использовании различных языков, а также то, как он использует свои языковые возможности, — все это относится к лингвистическому уровню взаимодействия.

Визуальный уровень конкретизирует отображение концептуальных объектов. Способ отображения концептуальных объектов, их взаиморасположение, возможности пользователя в управлении этим отображением, удобство восприятия и т.п. — все это относится к визуальным аспектам. Концептуальный, лингвистический и визуальный уровни относятся к двум уровням программных средств среды (уровню операционных систем и уровню программных средств общего назначения).

Функциональный уровень рассматривает взаимодействие пользователя с системой при решении прикладных задач, то есть типы воздействия пользователя на систему через функции пользовательского интерфейса и способы представления результатов в ответ на эти воздействия. Функциональный уровень относится к верхнему уровню концептуальной модели, на котором представлены прикладные программы ИС.

Вторая модель, реализующая графические пользовательские интерфейсы, представляет собой многоуровневую совокупность компонентов, в которой верхний уровень занимают прикладные программы, а нижние уровни — операционная система и процессор (рис.4.2). Модель начинается с компонентов связи с прикладными программами, содержащих множество точек, через которые пользователь контактирует с системой и элементами, такими как меню и иконы. Основные уровни модели не предназначены для выполнения функциональных или программных операций. Они используются только для выбора графических интерфейсов. В дополнении к прикладным программам, операционной системе и процессору, эта модель между ними имеет ***пять базовых уровней.***

Объектная модель располагается на высшем уровне GUI, но может быть не единственной. Объектная модель отражает реакции и взаимодействие приложений с внешней средой и между собой. Она определяет характер реализуемых прикладных функций, их использова-

ние, управление и построение объектов.

Прикладные программные интерфейсы (API) выполняют программно-языковые функции для связи пользовательских приложений с GUI. Программист может специфицировать функции (окна, меню, процессы отображения, иконы), которые необходимы для соответствующих прикладных программ. API включает средства, используемые разработчиками программ при создании GUI для конкретных приложений.

Ядро графического пользовательского интерфейса сосредоточивает экранные функции и элементы. В таких системах, как OpenLook и Motif— это полное средство обеспечения пользовательского интерфейса. В системе Microsoft Windows в ядре содержится только часть функций **GUI.** ***Оконная система*** выделяется в некоторых случаях для повышения гибкости развития всего GUI. Так, например, X Windows первоначально создавалась только как оконная система, а не весь графический пользовательский интерфейс. Однако системы функционально быстро развиваются, и оконные системы сливаются с ядром GUI.

Модели изображения формируются для различных пользовательских интерфейсов с помощью графических средств, в том числе X Windows. Для унификации интерфейсов могут применяться международные стандарты GKS (ISO 7942), PHIGS (ISO 9592) и другие.

Реализации представленной этой многоуровневой модели графических пользовательских интерфейсов весьма разнообразны и, в той или иной степени, ориентированы на аппаратные и операционные платформы. Однако общие тенденции состоят в стремлении обеспечить их,

по возможности, широкую мобильность между платформами с учетом применения международных стандартов и некоторых стандартов де-факто. При этом важнейшей задачей остается сохранение всей совокупности функций, доступных пользователю, унификация технологии и процедур его взаимодействия с приложениями при любых платформах.

Третья модель отражает общий методологический подход к реализации интерфейсов пользователя и выбору объектов стандартизации. Все компоненты пользовательского интерфейса должны поддерживаться операционными системами станций клиентов и серверов. Эта модель интерфейсов пользователя подразумевает разделение функциональных компонентов приложений на клиентские и сервер-

ные части независимо от какой-либо определенной архитектуры среды распределенной обработки данных.

В распределенных информационных системах с архитектурой клиент-сервер пользователи непосредственно взаимодействуют с клиентской частью системы, управляющей запуском и режимами работы прикладных программ. Серверные части прикладных программ обеспечивают доступ к данным и вычислительным ресурсам серверов. Система предоставляет пользователю формы документов и наборы процедур, которые он может выполнять. Эти функции обеспечиваются прикладными программами. Для реализации такого взаимодействия компоненты клиентской части среды, относящиеся к группе функций пользовательского интерфейса, обеспечивают средства работы с документами (текстами), механизмы управления окнами, готовые примитивы символов (алфавитно-цифровые и графические примитивы) для формирования нужных объектов, непосредственное манипулирование объектами на экране. Наконец, физический уровень взаимодействия пользователя с системой обеспечивают устройства ввода-вывода информации, входящие в состав автоматизированных рабочих мест. Таким образом, средства, поддерживающие функции интерфейсов пользователя, размещаются на трех уровнях ИС: уровне прикладных программ, уровне программных средств среды, уровне технических средств среды.

Система международных стандартов графических пользовательских интерфейсов. Особенностью пользовательского интерфейса как архитектурного компонента среды ИС является необходимость высокой согласованности всех функциональных элементов интерфейса. В связи с этим организации по стандартизации и промышленные консорциумы разрабатывают спецификации пользовательского интерфейса в виде гармонизированных профилей на основе базовых стандартов. Однако, на настоящий момент не существует подобного рода спецификации, которая охватывала бы все аспекты пользовательского интерфейса. Традиционно такие профили описывают либо структуру графического оконного интерфейса в целом, либо архитектуру построения распределенных приложений. Для построения профилей интерфейсов пользователя должны использоваться базовые стандарты, относящиеся к следующим ***классам объектов стандартизации:***

— типовые формы документов и процедуры работы с ними;

— элементы взаимодействия пользователя с алфавитно-цифровым интерфейсом;

— элементы взаимодействия пользователя с графическим интерфейсом;

— структура распределенных приложений в части средств, поддерживающих интерфейсы пользователя.

Типовые формы документов и процедуры работы с ними, рассматриваемые как объекты стандартизации, относятся к функциональному уровню взаимодействия пользователей с информационными системами. Объектами стандартизации, соответствующими процедурам из этого перечня являются элементы интерфейса пользователя, определяющие возможность начала соответствующей операции, ее ход и результат. Должна быть предусмотрена идентификация ошибочных действий и стандартизирована форма сообщения об ошибках. В связи с жестким регламентом выполнения процедур, средства настройки интерфейса пользователя на большинстве АРМов этих систем должны быть весьма ограничены либо отсутствовать.

Для информационно-аналитических и административно-вспомогательных систем в настоящее время типовые процедуры не определены. ***В этих документах должны быть описаны:***

— соответствие между элементами интерфейсов пользователя (экранными формами) и типовыми процедурами;

— последовательность допустимых операций и переходы между экранными формами;

— форма идентификации ошибочных действий и/или ситуаций;

— формы входных и выходных документов.

Большинство приложений, использующих алфавитно-цифровой интерфейс, предоставляют либо вариант интерфейса, разработанного специально в составе приложения, либо интерфейс, который может быть сконструирован при помощи средства разработки этого приложения. Существуют фирменные стандарты, определяющие построение алфавитно-цифровых интерфейсов.

Деятельность в области графических систем возглавляется и координируется 24-м подкомитетом ISO/IEC JTC1/SC24 при активном участии NIST, IEEE и ряда крупных программистских фирм. При создании графических стандартов используется принцип виртуальных ресурсов и разделения графической системы на несколько уровней (см. выше). Уровни могут использовать информацию нижних уровней и взаимодействовать с другими подсистемами посредством

стандартизированных программных интерфейсов, а также с помощью стандартизированных файлов или протоколов. Для этого выделены ***три направления стандартизации:***

— базисные графические системы;

— интерфейсы виртуальных устройств;

— форматы обмена графическими данными.

Стандарты базисных систем направлены на унификацию визуализации графических объектов. Интерфейсы виртуальных устройств разделяют аппаратно-зависимую и аппаратно-независимую части графических систем. При этом важную роль сыграло развитие растровых дисплеев и принтеров. Первоначально растровые файлы содержали только статические изображения. Появились проекты стандартов на форматы динамических (анимационных) изображений. На международном уровне упорядочена система непосредственного графического взаимодействия пользователей с базовыми средствами манипулирования графическими образами, которые стандартизированы де-юре ISO:

1. **ISO 9040:1990.** СОИ. ВОС. Услуги виртуального терминала. Базовый класс.
2. **ISO 9041:1990.** СОИ. ВОС. Протокол виртуального терминала. Базовый класс.
3. **ISO 7942:1985.** СОИ. МГ. Функциональное описание ядра графической системы (GKS). Приложение 1:1991.
4. **ISO 8651-1-4:1988.** СОИ. МГ. Языковые связи ядра графической системы (GKS). Ч.1:Фортран. Ч.2:Паскаль. Ч.3: Ада.Ч.4: Си.
5. **ISO 8805:1988.** СОИ. МГ. Функциональное описание трехмерного ядра графической системы (GKS-3D).
6. **ISO 8806-1-4:1991.** СОИ. МГ. Языковые связи трехмерного ядра графической системы (GKS-3D). 4.1: Фортран. 4.2: Паскаль. 4.3: Ада. 4.4: Си.
7. **ISO 8632-1-4:1992.** СОИ. МГ. Метафайл для хранения и передачи информации, описывающей изображение (CGM). 4.1: Функциональные требования. Дополнение 1:1990. 4.2: Символьное кодирование. Дополнение 1:1990. 4.3: Двоичное кодирование. Дополнение 1:1990. 4.4: Кодирование открытого текста. Дополнение 1:1990.
8. **ISO 9281-1,2:1990.** ИТ. Методы кодирования изображения. 4.1: Обозначение. 4.2: Процедуры для регистрации.
9. **ISO 9282-1:1988.** ИТ. Кодированные представления изобра-

жений. 4.1: Принципы кодирования для представления изображения в 7- и 8-битной среде.

10. **ISO 9592-1-3:1989.** СОИ. МГ. Иерархическая интерактивная графическая система программиста (PHIGS). 4.1: Функциональное описание. 4.2: Формат архивного файла. 4.3: Открытотекстовое кодирование архивного файла.
11. **ISO 9593-1-4:1990.** СОИ. МГ. Языковые связи иерархической интерактивной графической системы программиста (PHIGS). 4.1: Фортран. 4.2: Расширенный Паскаль. 4. 3: Ада. 4.4: Си.
12. **ISO 9636-1-6:1991.** ИТ. МГ. Методы сопряжения для диалогов с графическими устройствами. Функциональные требования. 4.1: Обзор, профили и согласованность. 4.2: Контроль. 4.3: Вывод. 4.4: Сегменты. 4.5: Ввод и отображение. 4.6: Растр.
13. **ISO 9638-1-4:1991.** ИТ. МГ. Методы интерфейса для диалогов с графическими устройствами. Языки библиотек CGI. 4.1: Фортран. 4.2: Паскаль. 4.3: Ада. 4.4: Си.
14. **ISO 9973:1988.** ТО. СОИ. Процедуры для регистрации графических элементов.
15. **IEEE-X Window —** стандарт де-факто. Многооконная графическая система.
16. **ISO 11072:1992.** ИТ. МГ. Компьютерная графика справочная модель.
17. **ISO 13719-1,2,3:1995.** Среда интеграции инструментальных средств (PCTE) на рабочих станциях — ориентированы на приложения разработчика программного обеспечения.

Услуги, регламентируемые представленной группой стандартов виртуальных терминалов и графики (GUI), определяют способы взаимодействия прикладных интерактивных систем, ориентированных на терминальную связь, в терминах передачи и манипулирования абстрагированными графическими образами. В базовой эталонной модели ВОС виртуальные терминалы и графика организуются на прикладном уровне и используют сервис представительского уровня. Базовый класс GUI организует образы, состоящие из блочно-символьных графических элементов, организованных в одно-, двух- / или трехмерные структуры. ***Услуги базового класса виртуальных терминалов позволяют*** [15,17,13]

— устанавливать соединение между двумя пользователями; -

— запрашивать и согласовывать подмножества услуг и сервисных параметров;

— передавать и манипулировать структурированными данными, независимо от способа внутреннего представления их информации, используемого каждым пользователем GUI;

— управлять целостностью взаимодействия с пользователем **GUI;**

— завершать взаимодействие пользователей согласованно и односторонне.

Стандарты GUI регламентируют ***процедуры управления диалогом*** на основе механизма права доступа и ряда правил упорядочения примитивов услуг. Для согласования параметров образуется ассоциация — соединение между прикладными объектами. Стандарты предусматривают операции двух типов: синхронные и асинхронные. Синхронные обеспечивают попеременный, взаимоисключающий обмен со стороны различных объектов, а асинхронные — назначают определенным объектам право доступа на все время их существования.

Семантика диалогов пользователей определяется в стандартах в терминах манипуляции над абстрактными объектами и типов самих объектов, к которым имеют доступ оба пользователя. Они совместно используют концептуальную область взаимодействия. Для описания операций сервиса GUI используются абстрактные объекты: концептуальная область взаимодействия и интерпретатор команд. Концептуальная область взаимодействия содержит память данных: одного или нескольких дисплейных объектов; управления, сигнализации и статуса; управления доступом; описаний дисплейных объектов, дополнительных устройств, объектов управления и т.п.

Для обеспечения мобильности прикладных программ и данных их взаимодействие с пользователями целесообразно организовывать по принципам и моделям, заложенным в международных стандартах графических систем. Однако развитие программных средств графических систем происходит очень динамично. В результате разработка стандартов де-юре в этой области не поспевает за реальной практикой совершенствования функциональных возможностей диалога пользователей и аппаратных графических устройств. Это приводит к появлению и массовому использованию в этой области стандартов де-факто [15,22,9]. Их формализация на международном уровне может быть не всегда целесообразна, так как ограничено время их существования вследствие интенсивного развития функций и изменения аппаратуры диалоговых средств.

Стандарт **ISO 9040** абстрактным образом определяет внешние услуги базового класса виртуального терминала (ВТ) внутри приклад-

ного уровня ВОС путем задания модели взаимодействия между пользователями этих услуг и связанных с ними событий, в зависимости от параметров, содержащихся в каждом примитиве и взаимосвязи последовательностей примитивов. Услуги обеспечиваются с использованием общего прикладного сервисного элемента, а также услуг уровня представления и могут использоваться пользователем. Услуги ВТ применяются для реализации интерактивных прикладных процессов, требующих обмена информацией с пользователем и ориентированных на терминал, т.е. для передачи и манипуляции графических образов, состоящих из дискретных графических элементов, организованных в одно-, двух- или трехмерную структуру, которые могут иметь атрибуты, задающие способ их отображения. Стандарт определяет синхронный и асинхронный режим работы, профили и функциональную среду ВТ. В приложении А определены два рекомендуемых профиля: для асинхронного и синхронного режима работы. В остальных приложениях содержатся некоторые профили ВТ, а также дополнительная информация о стандарте.

Стандарт **ISO 9041** конкретизирует процедуры передачи данных и управляющей информации, ориентированные на соединение, между протокольными автоматами, реализующими функции поставщика услуг базового класса виртуального терминала (ВТ), определенные в ISO 9040. Процедуры определены в терминах: взаимодействий между протокольными автоматами, осуществляемых элементами ВТ, и взаимодействий протокольного автомата с поставщиком услуг уровня представления путем обмена сервисными примитивами. Определены две категории систем ВТ: простые и способные к форматированию. Обе разновидности систем обеспечивают позначное взаимодействие пользователей, однако различаются возможностями удаленной системы обрабатывать данные, вводимые пользователями. Установлена структура протокольного блока данных, определены три подмножества процедур и способы согласования необходимого подмножества. Заданы аттестационные требования. В обязательных приложениях приведены таблицы переходов состояний для протокольного автомата ВТ и дан список стандартизованных имен, идентифицирующих объекты данного стандарта. Информационного приложение содержит примеры кодирования протокольных блоков данных в ASNI.

Стандарт **ISO 7942** определяет набор функций для программирования машинной графики — Систему графического ядра (GKS). GKS [16] является основой графической системы для приложений, кото-

рые создают двумерное машинное изображение на устройствах вывода построчной или растровой графики. Она поддерживает вывод и интерактивный режим путем обеспечения основных функций вывода и сегментации изображения, позволяет сохранять и динамически модифицировать изображение. Основой GKS является концепция графической рабочей станции, являющейся абстракцией физического устройства и состоящей из ряда устройств ввода и единственного устройства вывода. Несколько станций могут использоваться одновременно. Функции управления GKS обеспечивают работу с несколькими логическими рабочими станциями. Для построения изображений в системе используется три системы координат: мировая; нормированная и система координат устройства. Прикладной программист использует мировую систему координат, GKS переводит данные в нормированную систему, а устройство вывода выдает их на отображение в системе координат устройства. Поддержан вывод с изменением атрибутов шести примитивов: ломаная линия; набор маркеров; заполненная область; текст; массив ячеек и обобщенный графический примитив. Стандарт охватывает конструкции и библиотечные вызовы двумерного изображения, в принципе, любого вида. Прикладные программы позволяют адаптировать режимы GKS на рабочую станцию для расширения ее возможностей. Стандарт включает функции, для хранения и передачи внешнего графического файла. GKS определяет языково-независимое ядро графической системы. Для включения в язык программирования, GKS имеет встроенный языковозависимый уровень. N 1ST лицензирован комплект аттестационных тестов для GKS и организована сертификация соответствия ему графических систем. Разработано несколько десятков пакетов, реализующих стандарт на различных языках программирования и на различных аппаратных платформах.

Стандарт **ISO 8651-1-4** из четырех частей формализует внесение в языки программирования Фортран, Паскаль, Ада, Си ядра графической системы (GKS), которое встраивается на языковозависимом уровне, подчиняясь требованиям данного языка.

Стандарт **ISO** 8805 определяет набор функций для программирования машинной графики трехмерного ядра графической системы (GKS-3D). GKS-3D является основной графической системой для приложений, создающих трехмерные изображения на устройствах. графического вывода.

Стандарт из четырех частей **ISO 8806-1-4** формализует внесение в

языки программирования Фортран, Паскаль, Ада, Си трехмерного ядра графической системы (GKS-3D), которое встраивается на языковом зависимом уровне, подчиняясь требованиям данного языка. Привязка графического стандарта к языку программирования определяет отображение абстрактных типов данных, используемых в описании стандарта, на реальные типы данных конкретного базового языка. Привязка определяет также представление абстрактных имен функций и списков параметров средствами конкретного языка программирования с учетом всех налагаемых ограничений.

Некоторые принципы проектирования GUI

При разработке проекта пользовательского интерфейса учитывается мнение художника, человеческий фактор и интуиция потенциального пользователя. Некоторые основные принципы построения оконных интерфейсов проясняются после длительной работы с ними. Вот несколько самых важных принципов.

1. Необходимо понять, в чем состоит работа пользователя. Как правило, проектировщики интерфейсов проводят анализ заданий для понимания сути работы пользователя. Наш подход к характеристике прецедентов приблизительно соответствует такому анализу заданий.

2. Нужно сделать так, чтобы клиент чувствовал, что он контролирует взаимодействие Интерфейс пользователя должен обеспечивать возможность отмены действий.

3. Желательно предоставить клиенту несколько вариантов завершения каждой операции, связанной с интерфейсом (наподобие закрытия окна или файла), и не обращать особого внимания на его ошибки.

4. Внимание пользователя привлекает верхний левый угол экрана. Поэтому самую важную информацию необходимо размещать именно там.

5. Нужно учитывать пространственное расположение элементов. Связанные друг с другом компоненты экрана следует размещать рядом, например в одной рамке

6. Необходимо обращать особое внимание на удобочитаемость и ясность элементов интерфейса (использовать понятные всем и простые слова!). Для передачи идей и понятий используйте активный залог.

7. Надо ограничивать количество используемых цветов. Из всего многообразия необходимо остановиться на нескольких. Чрезмер-

ное множество цветов будет отвлекать пользователя от выполняемых им задач. Кстати, очень неплохо предоставить клиенту возможность самому изменять цвета.

8. Тем, кто подумывает об использовании цвета для выделения содержания, необходимо помнить, что пользователю не всегда легко провести ассоциацию между цветом и содержанием. Также следует знать, что некоторые пользователи (около 10% взрослых мужчин) плохо различают цвета, поэтому им будет сложно отличить один цвет от другого.

9. Как и в случае с цветами, необходимо ограничивать число шрифтов. Желательно избегать курсивов и витиеватых шрифтов. Шрифт под названием "Haettenschweiler" довольно приятен, но не стоит его использовать постоянно.

10. Старайтесь создавать компоненты (типа кнопок и списков) одинакового размера. При использовании компонентов различных размеров, разнообразии цветов и шрифтов создается мешанина или даже мозаика, которую специалисты в области GUI называют дизайном в стиле "клоунских штанов".

11. Выравнивайте компоненты и поля данных по левому краю. Это уменьшает нагрузку на глаза при просмотре экрана.

12. Если пользователь после прочтения и обработки определенного блока информации должен щелкать на кнопках, то такие кнопки лучше разместить справа от блока информации или же под этим блоком и, опять таки, справа. Это соответствует естественной тенденции (присущей нашей культуре) читать слева направо. Если одна из кнопок является кнопкой по умолчанию, то ее нужно выделить и сделать первой кнопкой в наборе.

Обучающие материалы

Прежде чем переходить к собственно рассмотрению обучающих материалов, необходимо оговорить одну вещь: эта книга не о том, как их делать. Создание хороших обучающих материалов является сложным и многогранным делом, требующим значительного опыта и дарования. Даже лишь пытаться впихнуть это занятие в несколько страниц, было бы непростительной поверхностностью. Так что в этой книге не будет рассказано, как писать или «подавать» справочную систему или документацию, будет только описано, как интегрировать их с интерфейсом. С другой стороны, если справочную систему или документацию обычно создают другие люди, то всплывающие под-

сказки, например, обычно пишут дизайнеры. Так что про дизайнерскую часть работы рассказано будет.

Что нам нужно и что у нас есть

Количество подсистем справки, нужных для того, чтобы пользователь научился пользоваться системой, довольно невелико, так что все их можно легко разобрать. Когда я говорю «подсистема справки» я имею в виду часть справочной системы (или документации), которая выполняет сугубо определенные функции и требует сугубо определенных методов представления.

Базовая справка объясняет пользователю сущность и назначение системы. Обычно должна сработать только один раз, объясняя пользователю, зачем система нужна. Как правило, не требуется для ПО, зато почти всегда требуется для сайтов.

Обзорная справка рекламирует пользователю функции системы. Также обычно срабатывает один раз. Нужна и ПО и сайтам, и нужна тем более, чем более функциональна система. Поскольку у зрелых систем функциональность обычно очень велика, невозможно добиться того, чтобы пользователи запоминали её за один раз. В этом случае оптимальным вариантом является слежение за действиями пользователя и показ коротких реклам типа «А вы знаете, что…» в случае заранее определенных действий пользователей (примером такого подхода являются помощники в последних версиях MS Office).

Справка предметной области отвечает на вопрос «Как сделать хорошо?». Поскольку от пользователей зачастую нельзя рассчитывать знания предметной области, необходимо снабжать их этим знанием на ходу. При этом действуют два правила: во-первых, пользователи ненавидят признавать, что они чего-либо не знают, соответственно, подавать это знание надо максимально «небрежным тоном»; во-вторых, наличие такого знания всегда повышает субъективную оценку справочной системы в целом, т.е. приводит к тому, что пользователи чаще обращаются к справочной системе и от этого эффективней учатся.

Процедурная справка отвечает на вопрос «Как это сделать?». В идеале она должна быть максимально более доступна, поскольку если пользователь не найдет нужную информацию быстро, он перестанет искать и так и не научится пользоваться функцией (возможно, никогда).

Контекстная справка отвечает на вопросы «Что это делает?» и «Зачем это нужно?». Как правило, наибольший интерес в ПО представляет первый вопрос, поскольку уже по названию элемента должно быть понятно его назначение (в противном случае его лучше вообще выкинуть), а в Интернете – второй (из-за невозможности предугадать, что именно будет на следующей странице). Поскольку пользователи обращаются к контекстной справке во время выполнения какого-либо действия, она ни в коем случае не должна прерывать это действие (чтобы не ломать контекст действий), её облик должен быть максимально сдержанным, а объем информации в ней – минимальным.

Справка состояния отвечает на вопрос «Что происходит в настоящий момент?». Поскольку она требуется именно что в настоящий момент, она не может быть вынесена из интерфейса. В целом это самая непроблематичная для разработчиков система справки, так что в этой книге разбираться она не будет.

Система должна индицировать все свои состояния

Сообщения об ошибках. Это тема настолько многогранная, что о ней мы поговорим отдельно несколько позже.

Поскольку наша цель состоит только в определении интеграции справочных систем с интерфейсом, разумно будет свести получившийся список типов обучающих материалов, которые нам нужны, со средами передачи этих материалов, которые у нас имеются. Среды же передачи, имеющиеся в нашем распоряжении, таковы.

Бумажная книга. На одном листе может быть сконденсировано очень много материала, легко позволяет читателю получить большой объем материала за один сеанс, наилучшим образом работает при последовательном чтении. Сравнительно плохой поиск нужных сведений. Объем практически всегда лимитирован.

Справочная карта. Отдельная краткая бумажная документация, демонстрирующая основные способы взаимодействия с системой (quick reference card). Будучи реализована на едином листе бумаги, позволяет пользователю повесить её перед собой. Хороша как средство обучения продвинутым способам взаимодействия с системой и устройству навигации в системе.

Структурированная электронная документация. Плохо предназначена для чтения больших объемов материала, зато обеспечивает легкий поиск и не имеет лимита объема. Занимает большой

объем пространства экрана. Плохо подходит для показа крупных изображений, зато в неё могут быть легко интегрированы видео и звук.

Фрагменты пространства интерфейса, показывающие справочную информацию. Занимают пространство экрана, но пространство ограниченное. Отвлекают внимание, как минимум один раз воспринимаются всеми пользователями. Как правило, неспособны передавать большой объем информации.

Всплывающие подсказки. Хорошо справляются с ответом на вопросы «Что это такое» и «Зачем это нужно», при условии, что объем ответов сравнительно невелик. Поскольку вызываются пользователями вручную, в обычном режиме не занимают пространства экрана и не отвлекают внимания пользователей. С другой стороны, очень легко вызывают отвыкание – после первого же случая неудовлетворения пользователя подсказкой, пользователь перестает вызывать и все остальные подсказки.

А теперь те виды справочных систем, за которые ответственен дизайнер интерфейсов, можно разобрать более детально.

Базовая и обзорная справки

Как уже было показано в главе «Почему пользователи учатся», объяснить пользователю сущность и назначение, а также отрекламировать функции системы чрезвычайно важно, поскольку только это создаёт желание учиться.

Эти системы справки обычно реализуются в бумажной документации. Это хорошо, но, вообще говоря, можно сделать и лучше, поскольку в последнее время появилась возможность интегрировать в справочную систему видео при помощи либо Macromedia Flash, либо Shockwave. Нет сомнений, что реклама, поданная не просто в виде текста с картинками, но в виде анимации, способна как повысить желание её просмотреть, так и повысить субъективное удовлетворение пользователей от системы. На этом уровне заниматься этим должен не дизайнер интерфейсов, а отдельный графический дизайнер. Как правило, работа эта не очень сложна (поскольку, фактически, всем содержимым этой анимации является показ сменяющих друг друга скриншотов). Сложно создать более-менее хороший сценарий (его лучше отдать профессиональному писателю). Дизайнер интерфейса в этом случае должен только составить список функций, которые нужно рекламировать.

Справка предметной области

Справка предметной области также реализуется обычно в бумажной документации, найти аргументы против такого положения вещей достаточно затруднительно. Однако, как минимум, часть её можно подавать пользователям в интерфейсе вместе с выдержками из обзорной справки (лучше динамически, a la «Помощник» из MS Office).

По-моему, справка предметной области является самой важной подсистемой справки. Достаточно упертый или «компьютерно-грамотный» пользователь сможет воспользоваться системой, лишенной всех справочных систем, более того, такой пользователь сможет даже *научиться* пользоваться такой системой. Но без знания предметной области он никогда не сможет пользоваться системой правильно и эффективно.

Процедурная справка

Лучшим местом для процедурной справки является выделенная справочная система. В неё, собственно говоря, она чаще всего и помещается. Вызывает, однако, сожаление тот факт, что разработчики чаще всего не привязывают темы справки к интерфейсу: когда пользователям непонятно, как выполнить нужное им действие, им приходится искать в справочной системе нужную тему. Это неправильно, тем более что технических проблем в этом нет.

Контекстная справка

Для контекстной справки заслуженно используют всплывающие подсказки (ToolTip) и, в последнее время, пузыри. Это очень правильное решение, с которым невозможно поспорить. Огорчает только практически полное отсутствие этого типа справки в интернете. Если разработчики ПО уже привыкли писать ко всем объектам и элементам управления подсказки, то для веб-дизайнеров это пока экзотика. Интересно при этом, что в интернете контекстная справочная система, как правило, более нужна, нежели в ПО – просто потому, что большинство сайтов являются однократно используемыми системами, пользователями которых являются изначально необученные люди.

Спиральность. В отличие от художественной литературы, справочные системы не предназначены для того, чтобы приносить удо-

вольствие, более того, поскольку пользователи обращаются к справочной системе при возникновении проблем, можно смело сказать, что использование справочной системы всегда воспринимается негативно. Таким образом, следует всемерно сокращать объем справочной системы, чтобы тем самым сократить длительность неудовольствия. К сожалению, сокращение объема не приводит к полному счастью, поскольку при малом объеме справочной системы возрастает риск того, что пользователи не найдут в ней ответы на свои вопросы. Куда ни кинь – всюду клин.

Есть, однако, исключительно эффективный метод решения этой проблемы: так называемые спиральные тексты. Идея заключается в следующем. При возникновении вопроса пользователь получает только чрезвычайно сжатый, но ограниченный ответ (1-3 предложения). Если ответ достаточен, пользователь волен вернуться к выполнению текущей задачи, тем самым длительность доступа к справочной системе (и неудовольствие) оказывается минимальной. Если ответ не удовлетворяет пользователя, пользователь может запросить более полный, но и более объемный ответ. Если и этот ответ недостаточен (что случается, разумеется, весьма редко), пользователь может обратиться к ещё более подробному ответу. Таким образом, при использовании этого метода, пользователи получают именно тот объем справочной системы, который им нужен. Спиральность текста считается нормой при разработке документаций. Есть веские основания считать, что она необходима вообще в любой справочной системе. Учитывая тот факт, что разработка спирали в справке непроблематична, я рекомендую делать её во всех случаях.

4. Субъективное удовлетворение

Натан Мирвольд, бывший вице-президент Microsoft, некогда высказал скандалезную по тем временам сентенцию «Крутота есть веская причина потратить деньги» (Cool is a powerful reason to spend money). Слово «Cool», которое я перевел как «Крутота», к сожалению, плохо переводится на русский язык, впрочем, засилье американской культуры привело к тому, что все мы неплохо представляем его смысл. Эта сентенция интересна, прежде всего, тем, что характеристика (cool), выбранная Мирвольдом, представляет собой просто таки триумф субъективности.

Предположение Мирвольда оправдалось. Исследования показали, что пользователи воспринимают одинаково положительно как убогие, но приятные интерфейсы, так и простые, эффективные, но сухие и скучные.

Таким образом, субъективные факторы имеют тот же вес, что и объективные. Разумеется, субъективность доминирует над объективностью только в тех случаях, когда покупателем системы выступает сам пользователь, но и в прочих случаях роль «крутоты» зачастую существенна, хотя бы потому, что повышение количества радости при прочих равных почти всегда приводит к повышению человеческой производительности. Это делает неактуальными вечные споры о первичности формы или функции. И то и другое важно.

Эстетика.

Все знают, что значительно легче и приятнее пользоваться эстетически привлекательными объектами. Это наблюдение породило весь промышленный дизайн, включая дизайн одежды, интерьеров и так далее. В то же время в другой области промышленного дизайна, а именно в дизайне интерфейсов, это наблюдение до сих пор как следует, не утвердилось: бои между пуристами (интерфейс должен быть, прежде всего, работоспособным) и маньеристами (красота – это страшная сила) никоим образом не затихают. В то же время «срединный путь» до сих пор не найден, интерфейсы, равно удобные и эстетически привлекательные, до сих пор существуют в единичных экземплярах. Происходит это преимущественно оттого, что компьютер до сих пор воспринимается всеми как нечто совершенно новое, не имеющее корней в до компьютерной реальности. Во многом это правильно. Кто бы что ни говорил, но массовые представления о прекрасном за последние сто лет не выросли. Логичные в таких условиях интерфейсы в эстетике художника Шишкина по меньшей степени противоестественны. С другой стороны, принципы многих направлений дизайна вполне применимы к дизайну интерфейсов, он имеет черты, как сближающие его с иными направлениями дизайна, так и разъединяющие. Разберем это подробнее.

- *Внимание к деталям*. Интерфейс состоит из отдельных деталей, каждая из которых действует сравнительно независимо, поскольку раскрывает различную функциональность. Это сближает дизайн интерфейса в целом с книжным дизайном, характерными, как раз пристальным вниманием к мелочам.

- *Интерфейс не самоценен*. Опять сближение с книжным дизайном (никто не покупает книгу из-за качества её верстки).

- *Интерфейс передает информацию своему пользователю*. Опять книжный дизайн и коммуникационный дизайн вообще. Фактически, плакат со схемой метро обладает явно выраженным интерфейсом, другой разговор, что этот интерфейс более однонаправленный, нежели двусторонний.

- *Интерфейс обычно предназначен для длительного использования*. Это серьезно отличает его от графического дизайна вообще (никто не будет рассматривать журнальный разворот часами), но зато сближает опять с книжным дизайном и дизайном среды обитания.

- *Интерфейс функционален*. Очень часто приходится искать компромисс между эстетикой и функцией. Более того, интерфейс сам по себе зарождается в функциональности, «интерфейс ни к чему» просто не может существовать. Это сближает дизайн интерфейса с промышленным дизайном.

- *Интерфейс готового продукта образуется не сам по себе, но в результате промышленного производства*. Дизайнер интерфейса не сам производит интерфейс – за него это делают программисты, имеющие свои ограничения (стоимость, технология и так далее).

Таким образом, принципы многих направлений дизайна вполне применимы к дизайну интерфейса, при этом донорами преимущественно выступают книжный, коммуникационный и промышленный дизайны. Итак, какие их принципы могут быть использованы в дизайне интерфейса?

Конструируемый предмет должен быть незаметен в процессе его использования. Он должен приятно ощущаться на бессознательном уровне Для этого:

- Избегайте развязности в изображении Лучше, чтобы он был скромнее. Во что бы то ни стало, добивайтесь того, чтобы интерфейс был неощущаем

- Избегайте ярких цветов. Существует очень немного цветов, обладающих и яркостью, и мягкостью (т.е. не бьющих по глазам). На экране их значительно меньше, поскольку в жизни такие цвета обычно моделируются как собственно цветом, так и текстурой, с чем на экране есть проблемы.

- Избегайте острых углов в изображении

- Старайтесь сделать изображение максимально более легким и воздушным.

- Старайтесь добиваться контраста не сменой насыщенности элементов, а расположением пустот.

На рис. 10 показан пример закономерностей в диалоговом окне. Старайтесь минимизировать количество констант (тем более, что двух констант обычно хватает на все). Разумеется, единожды примененных закономерностей необходимо придерживаться во всей системе.

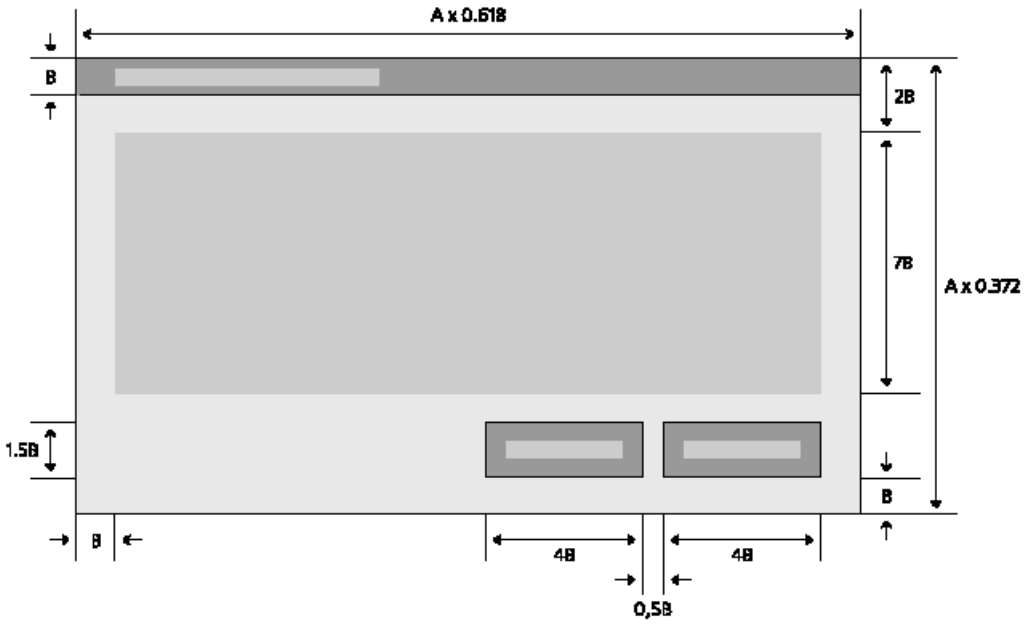

Рис. 10. Визуальный дизайн: использование компонентов

Вы должны правильно использовать компоненты визуального дизайна, чтобы показать пользователю для чего нужно каждое окно, и как им пользоваться. Хорошо выполненный дизайн выглядит чистым, простым и аккуратным. Его можно понять одним взглядом. Пользователь должен сразу распознавать, какие данные можно редактировать, какие нет; по каким объектам можно щелкать мышью и какие объекты можно перетаскивать.

Чтобы выполнить этот принцип, вы должны понимать, как пользователь воспринимает различные элементы управления. Существует понятие "предоставленная возможность" (affordance), хотя сам термин выглядит довольно неуклюже. "Предоставленные возможности" это визуальные характеристики объектов, которые сигнализируют о том, что с ними можно сделать. Например гребни на крышке банки с

овощами говорят мне о том, что крышка отвинчивается, а не открывается открывашкой. Поля ввода, например, приглашают пользователя ввести любое текстовое или числовое значение. Если же набор вводимых величин ограничен, то лучше использовать выпадающий список.

Вы также должны понимать принципы визуальных сообщений. Размер, цвет, яркость, местоположение, форма и текстура - все это средства, которые используются для того чтобы сгруппировать элементы вместе по важности или по похожести. Если у вас есть возможность воспользоваться услугами профессионального визуального дизайнера, не пренебрегайте ею.

Если вы пропустите эту стадию, вы рискуете получить программу, которая выглядит непрофессионально и непривлекательно. Если вы будете пренебрегать визуальным дизайном, ваша программа будет посылать хаотические сигналы, что приведет к увеличению ошибок, путанице и раздражению пользователя

Красота понятие относительное. Для одних красивыми могут считаться только живописные закаты, для других картины художника Кустодиева, а для третьих – комбинация вареных сосисок, зеленого горошка и запотевшей бутылки пива. Это делает красоту вещью не слишком универсальной. Хуже того. Любая красота со временем надоедает и в лучшем случае перестает восприниматься. Именно поэтому в интерфейсах обычно не место красоте. Элегантность и гармония гораздо лучше. Во-первых, они не надоедают. Во-вторых, они редко осознается потребителями, обеспечивая неощущаемость. В-третьих – они приносят эстетическое удовольствие независимо от культурного уровня потребителя (так, древнегреческие и слегка менее древние римские здания воспринимаются нами красивыми, несмотря на абсолютную разницу культур и времени). В-четвертых, в производстве они гораздо удобнее красоты, поскольку сравнительно легко ставятся на поток. Итак, каким образом надо действовать, чтобы добиться элегантности:

- *Старайтесь сделать интерфейс максимально насыщенным визуальными закономерностями*. Есть универсальное правило – чем больше закономерностей, тем больше гармонии. Даже самые незначительные закономерности всё равно воспринимаются. Под законо-

мерностью я понимаю любое методически выдерживаемое соответствие свойств у разных объектов, например, высота кнопок может быть равна удвоенному значению полей диалогового окна.

Стремитесь не столько к красоте интерфейса, сколько к его элегантности

- *Всемерно старайтесь использовать модульные сетки*, т.е. привязывайте все объекты к линиям (лучше узлам) воображаемой сетки, которую выдерживайте во всем интерфейсе.
- *Старайтесь привязывать все размеры и координаты* (как минимум пропорции диалоговых окон) к золотому сечению (0.618 х 0.382).

Объективное и субъективное ощущение времени.

Любой человек хочет работать быстро. Если работу (или, понимая шире, любое действие) можно выполнить быстро, у человека возникает приятное ощущение. Хитрость тут в том, что *субъективное ощущение времени* зачастую сильно отличается от *объективного,* так что методы повышения реальной скорости работы, описанные в начале книги, помогают отнюдь не всегда.

Человеческое восприятие времени устроено своеобразно. С одной стороны, пользователи способны обнаружить всего 8-процентное изменение длительности в двух или четырехсекундном времени реакции системы. С другой стороны – не могут точно определить суммарную длительность нескольких последовательных действий. Более того, воспринимаемая продолжительность действий напрямую зависит от уровня активности пользователя, так что субъективная длительность последовательности действий всегда ниже такой же по времени паузы. Это наблюдение вовсе не результат напряженных исследований: все знают, что при бездействии (скуке) время течет невыносимо медленно. Важно понимать, что действие может быть не обязательно физическим: лежа на диване и смотря фильм, т.е. не совершая почти никаких физических действий, время можно потратить очень быстро.

Таким образом, *субъективную скорость работы* можно повысить двумя способами:

- *Заполнение пауз между событиями*. Есть данные о том, что если в периоды ожидания реакции системы пользователям показывается индикатор степени выполнения, субъективная продолжитель-

ность паузы существенно снижается. Судя по всему, чем больше информации предъявляется пользователям в паузах, тем меньше субъективное время. С другой стороны, эта информация может вызвать стресс в кратковременной памяти, так что пользоваться этим методом надо осторожно.

- *Разделение крупных действий пользователей на более мелкие*. При этом количество работы увеличивается, но зато субъективная длительность снижается. Плох этот метод тем, что увеличивает усталость.

С другой стороны, повышение объективной скорости работы зачастую способно повысить и субъективную скорость. Другой разговор, что в каждом конкретном случае это нужно проверять секундомером. В этой проверке нет ничего сложного: нужно просто сравнить объективную длительность действия с его субъективной длительностью.

По острию ножа. Нет ничего более неприятного, чем психологическое напряжение, иначе говоря – стресс. Оператор на атомной станции или пилот самолета не просто спокойно сидят и нажимают на кнопочки, но нажимают на кнопочки, зная, что если они нажмут не на ту кнопочку, всем придется очень и очень туго. Большинство компьютерных программ и сайтов не требует от пользователя такой степени психологического напряжения. Тем не менее, им есть, куда расти.

Дело в том, что почти всё время пользователь может что-либо испортить и знает это. Он может отформатировать жесткий диск, может стереть или испортить нужный файл. Неудивительно, что пользователь часто боится. А если не боится, то склонен недооценивать свои возможности к разрушению, отчего снижается бдительность. Куда ни кинь, всюду клин. Единственным полноценным решением является возможность отмены пользователем своих предыдущих действий, без ограничения количества уровней отмены и типа отменяемых действий. Задача эта непростая, но зато результат крайне существенен. Пользователь, знающий, что он *не может* совершить ошибку, испытывает радость и умиротворение. К сожалению, создание таких систем, не будучи исключительно трудным делом с точки зрения технологии (мы, как никак, живем уже в двадцать первом веке) требует смены парадигмы мышления программистов, так ожидать скорого наступления эры всеобщего счастья не приходится. Зачастую

более реалистичным решением является давно уже существующая практика прятать опасные для пользователя места интерфейса. Формально, это хороший и правильный способ. Проблема заключается в том, что при этом логично прятать *все* функции, изменяющие данные, например банальная функция автоматической замены может мгновенно уничтожить текст документа (достаточно массовидно заменить одну букву на любую другую и забыть отменить это действие).

Другим фактором, существенно влияющим на субъективное удовлетворение пользователей, является чувство контроля над системой. Существует значительная часть пользователей, для которой использование компьютера не является действием привычным. Для таких пользователей ощущение того, что они не способны контролировать работу компьютера, является сильнейшим источником стресса. Для остальных пользователей отсутствие чувства контроля не приносит стресса, но всё равно приводит к неудовольствию.

Таким образом, пользователей нужно всемерно снабжать ощущением, что ничего не может произойти, пока этого не захочется самому пользователю. Функции, работающие в автоматическом режиме, но время от времени просыпающиеся и требующие от пользователей реакции, вызывают стресс. В любом случае, стоит всеми силами внушать пользователям мысль, что только явно выраженное действие приводит к ответному действию системы (это, в частности, главный аргумент против ролловеров – пользователь ещё ничего не нажал, а уже что-то произошло).

Вон отсюда, идиот!

Ни один пользователь не может долго и продуктивно работать с системой, которая его огорчает и обижает. Тем не менее, такие «скандальные» системы являются нормой. Виной тому сообщения об ошибках. Обратите внимание, что здесь я пишу не о том, как предупреждать ошибки пользователя, но о том, почему сообщения об ошибках плохи.

Дело в том, что большинство сообщений об ошибках в действительности не являются собственно сообщениями об ошибках. На самом деле они показывают пользователю, что система, которой он пользуется:

- недостаточно гибка, чтобы приспособиться к его действиям
- недостаточно умна, чтобы показать ему его возможные границы его действия

- самоуверенна и считает, что пользователь дурак, которым можно и нужно помыкать.

Разберем это подробнее.

Недостаточная гибкость. Многие программы искренне «уверены», что пользователь царь и бог, и когда оказывается, что пользователь хочет невозможного (с их точки зрения), они начинают «чувствовать» разочарование. Проявляют же они свое чувство диалогами об ошибках.

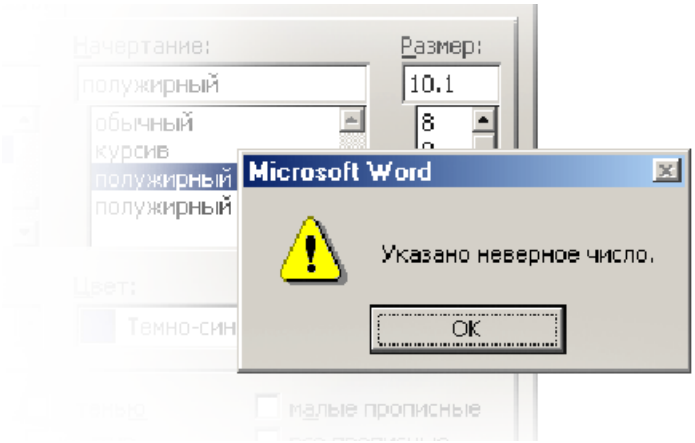

Рис. 11. Вот что бывает, если пользователь попытается ввести значение, которое ему нужно, но которое система не умеет обрабатывать. Тут возможно три альтернативных решения. Во-первых, при проектировании системы можно более тщательно подходить к выбору её функциональности. Во-вторых, можно просто игнорировать неправильность значения, округляя его до ближайшего возможного (индицируя, возможно, самостоятельность действий системы однократным миганием соответствующего полявводa). В-третьих, вместо обычного поля ввода можно использовать крутилку

В действительности множество действий пользователя направлены не на то, чтобы сделать так, а не иначе, а на то, чтобы сделать *примерно* так, как хочется. Пользователю часто нет дела, можно добиться точного результата или нет. Показывать ему в таких случаях диалог об ошибке глупо, поскольку пользователю не нужно ничего знать. Ему нужен результат.

Нежелание показать границы действия. Во многих случаях пользователь совершает действия, которые воспринимаются про-

граммой как неправильные, не потому, что он дурак, но потому, что система не показала ему границ возможного действия.

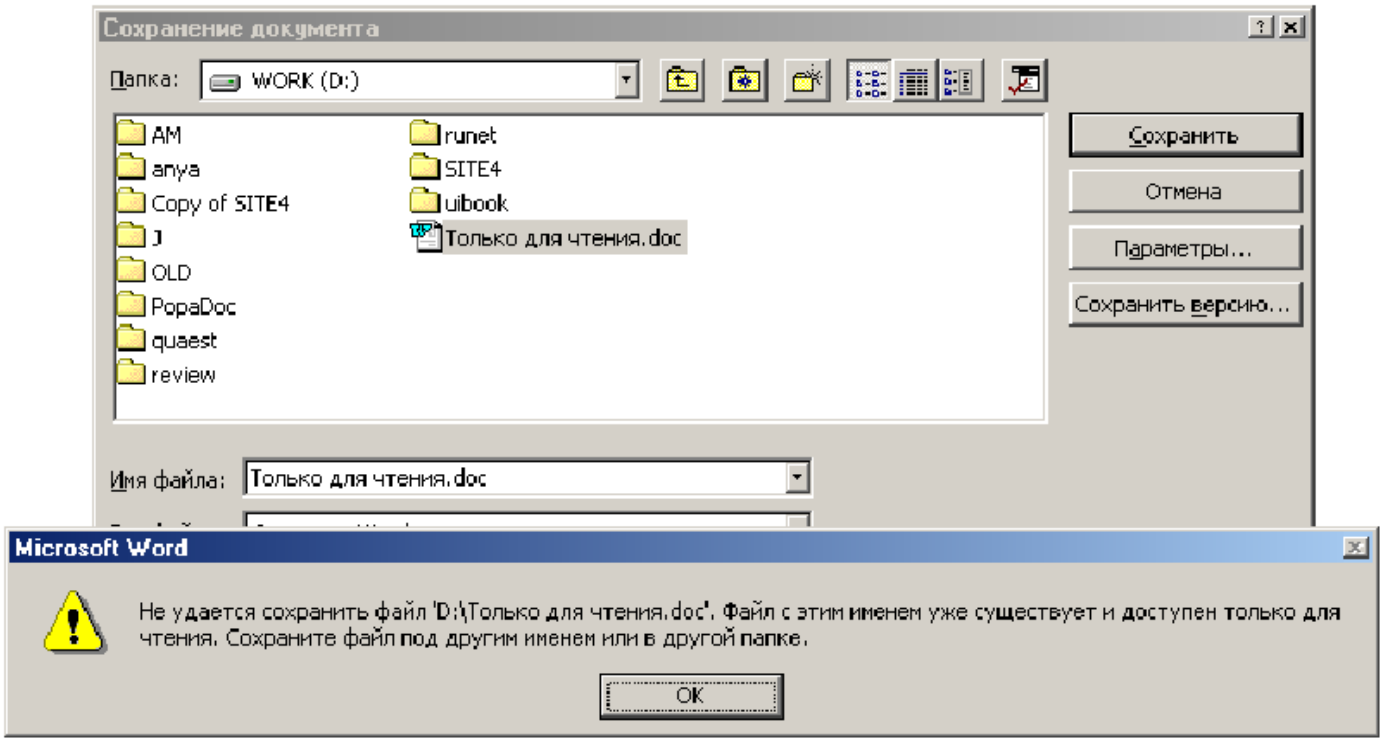

Рис. 12. Типичное сообщение об ошибке, вызванное нежеланием системы показать пользователю границы его действий. С одной стороны, оно разумно – файл не может быть записан под этим именем. С другой стороны, это сообщение показывается пользователю каждый раз при попытке перезаписать такой файл. Если бы названия всех защищенных от записи файлов отображались бы не черными, но серыми, это сообщение пришлось бы показывать пользователю только один раз в его жизни. © Microsoft.

Все такие сообщения порочны, поскольку их можно было бы избежать, просто блокировав возможность совершения некорректных действий или показав пользователю их некорректность до совершения действия.

Самоуверенность. Нормой также являются случаи, когда система пытается выставить дело так, как будто пользователь идиот, а система, наоборот, есть воплощение безошибочности и правоты.

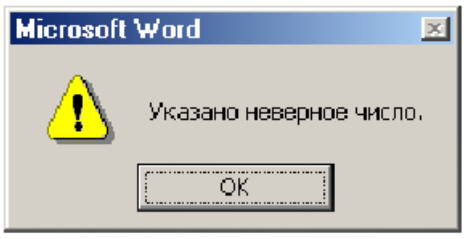

Рис. 13. Для кого неверное? И кто, собственно, виноват, система или пользователь?

В действительности не пользователь сделан для системы, но система для пользователя. Таким образом, как-либо ущемлять пользователя неправильно.

Пользователи ненавидят сообщения об ошибках

Суммируя, можно сказать, что почти любое сообщение об ошибке есть признак того, что система спроектирована плохо. Всегда можно сделать так, чтобы показывать сообщение было бы не нужно. Более того. Любое сообщение об ошибке говорит пользователю, что он дурак. Именно поэтому пользователи не любят сообщения об ошибках, а если говорить откровеннее, их ненавидят.

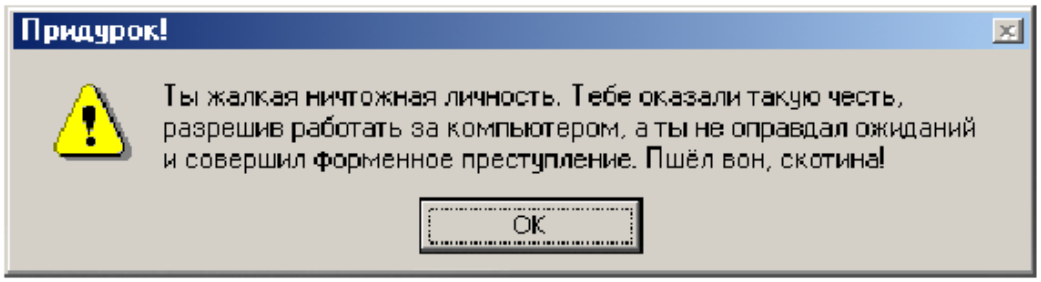

Рис. 14. Именно так пользователи воспринимают любые сообщения об ошибках.

Таким образом, почти все сообщения об ошибках должны быть удалены. Разумеется, речь идет не о том, чтобы просто выкинуть куски кода из программы, а о том, что системы изначально надо проектировать так, чтобы в них отсутствовала необходимость в таких сообщениях. Невозможно полноценно работать с системой, которая по нескольку раз за день тебя ругает.

Каким должно быть сообщение об ошибке

Теперь можно рассказать, каким должно быть сообщение об ошибке, тем более, что ничего сложного в создании идеального со-

общения нет. Напротив, всё очень просто. Идеальное сообщение об ошибке должно отвечать всего на три вопроса:

- В чем заключается проблема?
- Как исправить эту проблему сейчас?
- Как сделать так, чтобы проблема не повторилась?

При этом отвечать на эти вопросы нужно возможно более вежливым и понятным пользователям языком. В качестве примера идеального сообщения об ошибке удобно взять какое-либо существующее сообщение (из тех, которые точно нельзя просто изъять из системы) и попытаться это сообщение улучшить. Например, попытаемся улучшить уже упоминавшееся в предыдущей главе сообщение о невозможности перезаписать заблокированный файл.

Итак, старое сообщение об ошибке гласило: «Не удается сохранить файл «D:\Только для чтения.doc». Файл с этим именем уже существует и доступен только для чтения. Сохраните файл под другим именем или в другой папке». Это довольно неплохое сообщение, во всяком случае, оно гораздо лучше, чем «Указано неверное число». Но и его можно улучшить. Сначала надо разобраться, в каких случаях оно появляется. Это несложно: оно может появляться, если пользователь попытался сохранить файл на компакт-диске, или же пытается сохранить файл, незадолго перед этим скопировав этот файл с компакт-диска. Случаи, когда файл заблокирован сознательно, в жизни редки, так что их чаще всего можно не учитывать. Главный враг – компакт-диск.

Тут возможно несколько не противоречащих друг другу решений. Во-первых, просто можно блокировать возможность что-либо записать на диске, запись на который невозможна. Собственно говоря, запись и так блокируется, но сообщением об ошибке. А можно просто не показывать диски, на которые нельзя записывать, в окне записи, что эффективнее, поскольку делает ошибку невозможной. Во-вторых, как уже говорилось, можно показывать файлы, защищенные от записи, иначе, чем файлы незащищенные. Это будет работать, но тоже неидеально. Что делать пользователю, который всё-таки хочет перезаписать файл? Сейчас в такой ситуации приходится записывать файл под новым именем, потом стирать старый, а новому давать имя старого. Это и потери времени и ошибочно стертые файлы (лучший способ сделать так, чтобы пользователи не стирали нужные файлы,

заключается в том, чтобы лишить пользователей необходимости вообще что-либо стирать в нормальном режиме работы).

Таким образом, сообщение об ошибке должно стать не только сообщением – оно должно позволять разблокировать файлы, разблокировать которые возможно (т.е. записанные не на компакт-диске). Таким образом, получается, что нужно сделать несколько изменений в интерфейсе:

- Диски, на которые ничего нельзя записать, не показываются в диалоговом окне сохранения файлов.
- Заблокированные файлы на остальных дисках показываются иначе, нежели файлы незаблокированные.
- При попытке записать документ поверх заблокированного, появляется сообщение об ошибке примерно такого вида:

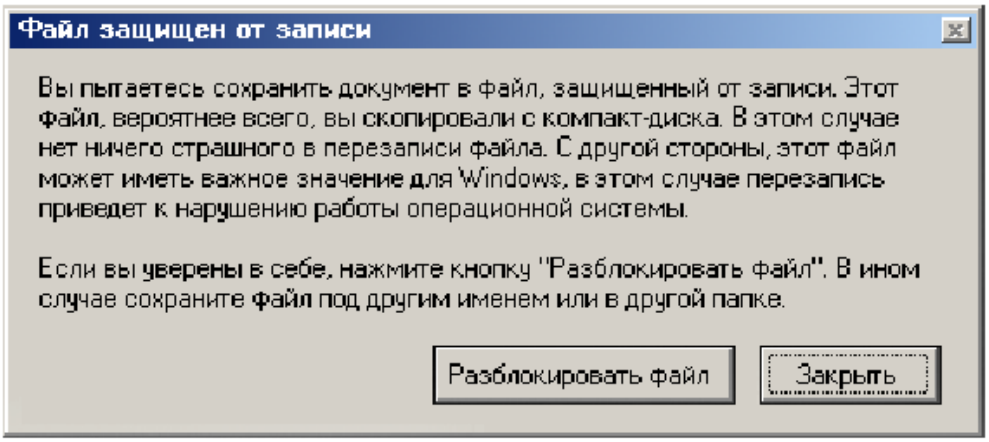

Рис. 15. Улучшенное сообщение об ошибке. Обратите внимание, что кнопка Закрыть выбрана по умолчанию, чтобы снизить вероятность перезаписи важных файлов. Конечно, лучше всего было бы, чтобы ОС сама снимала с копируемых с компакт-диска файлов метку Read Only. Многие проблемы при этом бы исчезли, поскольку защищенными от записи остались только действительно важные для ОС файлы.

Про этот пример осталось сказать немного. Во-первых, никогда не забывайте показывать текст сообщений об ошибке техническому писателю. Во-вторых, всемерно старайтесь делать текст сообщения возможно более коротким. В-третьих, диалоговое окно не самый лучший способ показывать сообщения об ошибках, во всяком случае, в ПО. Дело в том, что в Windows появился элемент управления, зна-

чительно лучше предназначенный для показа сообщений. Называется этот элемент весьма поэтично: пузырь

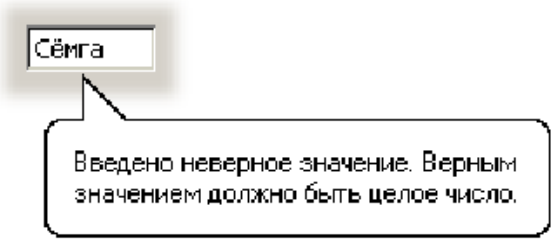

Рис. 16. Пузырь в его лучшем проявлении (не обращайте внимания на текст).

Пузырь, по сравнению с диалоговым окном, имеет существенные достоинства. Во-первых, он гораздо слабее сбивает фокус внимания, нежели окно. Во-вторых, чтобы закрыть пузырь, пользователям не обязательно нажимать на какую-либо кнопку, достаточно щелкнуть мышью в любом месте экрана. В-третьих, он не перекрывает значимую область системы. В-четвертых, что самое главное, он показывает, в каком именно элементе управления была допущена ошибка. Все это делает пузырь вещью совершенно незаменимой. Я уверен, что через пару лет 80 процентов всех сообщений об ошибках будет появляться в пузырях.

К сожалению, пузыри имеют и недостатки. Во-первых, в них не может быть никаких элементов управления, так что использовать пузырь в предыдущем примере, например, было невозможно. В Windows пузыри вообще реализованы довольно бедно. Во-вторых, пузырей вообще нет в интернете. Так что правило тут простое – используйте пузыри всегда, когда это возможно, и рыдайте, когда их нет.

Пароли. Не только пароли, но сама по себе идея секретности вызывает у пользователей изжогу. Причем, что нечасто случается, изжогу пароли вызывают не только у какой-то отдельной группы пользователей, такой как новички, а у всех. Объясняется это просто – соблюдение секретности требует от пользователей выполнения действий, не приносящих пользы, но уменьшающих *потенциальный* вред. Человеческая же природа такова, что потенциальное кажется неважным, а вот действия, которые приходится совершать, важны чрезвычайно. В результате пароли не любит никто.

Разумеется, это не совсем интерфейсная проблема, скорее политическая, но она оказывает такое влияние на результат, что обойти её невозможно. Итак, пароли. Они имеют три принципиальных проблемы. Во-первых, как уже было сказано, пользователи не любят их вводить. Во-вторых, пользователи либо забывают пароли, либо пароли не работают, т.е. ничего не защищают, поскольку пользователи используют те пароли, которые они помнят и которые, соответственно, непроблематично подобрать. В-третьих, в интернете часто происходит так, что пользователи, не желая вводить пароль (и регистрироваться, к слову говоря) так никогда не попадают в главную часть сайта. В результате, пароли есть явное и несомненное зло, вопрос состоит лишь в том, как это зло уменьшить.

Для этого есть несколько путей. Во-первых, от них можно отказаться вообще. Этот путь почти всегда предпочтителен, проблема в том, что он вызывает существенное отторжение у сетевых администраторов (которые все как один параноики). В самом деле, зачем требовать от пользователя пароль, если подобрать этот пароль злоумышленник сможет без труда?

И обратно: поскольку единственным способом создания действительно трудноподбираемых паролей является генератор случайных чисел, каковые числа (символы, по необходимости) невозможно запомнить, пользователи либо будут записывать их на бумажке (в этом случае пароли еще легче украсть), либо будут забывать, что дорого стоит. Я, например, вынужден пользоваться четырьмя действительно случайными паролями, причем тремя из них не реже раза в неделю. При этом, несмотря на то, что я пользуюсь этими паролями не менее года, я не помню *ни одного*. В результате, все четыре пароля записаны у меня на бумажке, которую увидеть может любой (я её не прячу). Какая уж тут защита?

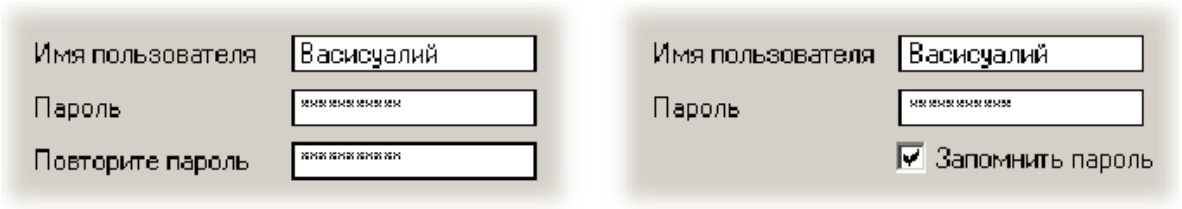

Рис. 17. Стандартный способ человеколюбивого использования паролей: слева во время регистрации, справа в дальнейшей работе.

Этот метод хорош, когда дело касается защиты информации. Однако гораздо чаще, особенно в интернете, важна не столько защита информации, сколько идентификация пользователя (которая может сопрягаться с защитой, а может и не сопрягаться). В этом случае от паролей отказаться невозможно, просто потому, что не существует другой, столь же эффективной, технологии идентификации. При этом содержимое пароля как таковое не очень важно, важно его отличие от паролей других пользователей (обратите внимание, что в этом случае само по себе *имя пользователя* является паролем). Обычно в таких условиях при регистрации применяют стандартную группу элементов «имя, пароль, подтверждение пароля», после чего, уже при нормальной деятельности, пользователь получает возможность ввести пароль только в первый раз (см. рис. 17).

Это неплохое решение, но и оно не без недостатка. Дело в том, что пользователь, однократно введя пароль, потом его забывает, поскольку из-за отсутствия повторения пароль никогда не попадает в долговременную память. В результате, когда пользователь переходит на другой компьютер или переустанавливает ОС, ему приходится регистрироваться снова, что нехорошо. Во-первых, эта лишняя работа раздражает. Во-вторых, база пользователей при этом разбавляется дубликатами, что всегда плохо сказывается на стоимости этой базы. В-третьих, что иногда самое главное, зарегистрировавшийся во второй раз пользователь лишается возможности получить уже использованное им ранее имя, что огорчительно: например, пользователь, забывший пароль от почтовой службы, лишается как доступа к своему почтовому ящику, так и возможность получить прежний почтовый адрес. Конечно, *некоторые* пользователи воспользуются тем или иным способом узнать свой прежний пароль, но отнюдь не все.

Это значит, что помимо какого-либо механизма напоминания пользователям их потерянных паролей (неважно какого, лишь бы заметного), очень полезно не скрывать от пользователей их пароль, т.е. выводить его в поле ввода не звездочками, а открытым текстом. Это решение хорошо ещё и тем, что количество ошибочно набранных паролей здорово сократится (тяжело набирать на клавиатуре невидимое).

Таким образом, эффективнее всего максимально здраво определить степень важности защищаемой информации, после чего выбрать адекватную схему защиты, стараясь, чтобы она была максимально

лёгкой для пользователей. Как правило, их субъективное удовлетворение важнее потенциального вреда от потери информации.

Самовыражение Страсть к самовыражению является одной из самых сильных черт человеческой натуры. В большинстве случаев люди самовыражаются через вещи (двигают мебель и покупают модную одежду), когда нет вещей – насвистывают или поют изящные песни «лишенным приятности голосом». Ни один человек не может существовать сколько-нибудь продолжительное время в обстановке, не допускающей самовыражения. Среднестатистический человек проводит в день около восемнадцати минут в туалете, этого времени достаточно, чтобы вызвать острое желание выразить себя: покупается особая, не такая как у других, вешалка для рулона туалетной бумаги, стены красятся в сравнительно уникальные цвета, а наиболее творческие «пользователи туалета» вешают на дверь зеркало и т.д.

Неудивительно, что пользователи хотят выразить себя и в программах, которыми они пользуются. Соответственно, возможность настроить систему под свои нужды является мощной причиной субъективного удовлетворения.

Проблема здесь в том, что это очень дорого стоит. Времени на самовыражение уходит крайне много. Во-первых, пользователи не склонны удовлетворяться первым получившимся вариантом (творчество процесс итерационный). Во-вторых, когда выходят новые версии системы (или имеющаяся версия разрушается от излишнего самовыражения), все приходится начинать сначала. По моим, очень грубым прикидкам, на такой процесс самовыражения в среднем уходит около 45 минут в неделю, если согласиться с этим числом, получается, что организация всего с сотней работников теряет на этом еженедельно 75 часов, что почти равняется потере недельной работы двух человек.

С другой стороны, настроенный под свои нужды интерфейс, судя по всему, снижает усталость работников и повышает их рабочее настроение.

В таких условиях потеря 45 минут в неделю может оказаться скомпенсированной благодаря повышению производительности труда. Таким образом, в продуктах, продаваемых пользователям напрямую, возможность настройки под конкретного пользователя обязательна. Во всех остальных случаях, судя по всему, нужен способ настройки, позволяющий максимально изменить вид системы минимумом команд (чтобы снизить время, затрачиваемое на самовыражение). Хорошим вариантом является банальный выбор варианта гото-

вой настройки из списка без возможности модифицировать встроенные варианты.