Интерфейсы автоматизированных систем

Кафедра «Вычислительные системы и информационная безопасность»

Лекционный курс

Автор

Ганжур М.А, Ганжур А.П.

Ростов-на-Дону,
2018

Аннотация

Лекционный курс предназначен для студентов очной, заочной форм обучения по направлениям 10.03.01 «Информационная безопасность», 09.03.02 «Информационные системы и технологии»

Автор

Ганжур Марина Александровна–

ст. преподаватель кафедры «Вычислительные системы и информационная безопасность»

Ганжур Алексей Петрович–

ст. преподаватель кафедры «Вычислительные системы и информационная безопасность»

ОГЛАВЛЕНИЕ

Тема 1. Общее представление об информационной системе

Рассмотрим, что представляет собой понятие *"информационная система".*

Специфика информационных программных систем

В зависимости от конкретной области применения информационные системы могут очень сильно различаться по своим функциям, архитектуре, реализации. Однако можно выделить, по крайней мере, **два свойства,** которые являются общими для всех информационных систем.

Во-первых, любая информационная система предназначена для сбора, хранения и обработки информации. Поэтому **в основе любой информационной системы лежит среда хранения и доступа к данным**. Среда должна обеспечивать уровень надежности хранения и эффективность доступа, которые соответствуют области применения информационной системы.

Заметим, что в вычислительных программных системах наличие такой среды не является обязательным. Основным требованием к программе, выполняющей численные расчеты (если, конечно, говорить о решении действительно серьезных задач), является ее быстродействие. Нужно, чтобы программа произвела достаточно точные результаты за установленное время.

Во-вторых, информационные системы ориентируются на конечного пользователя, например, банковского клерка. Такие пользователи могут быть очень далеки от мира компьютеров. Для них терминал, персональный компьютер или рабочая станция представляют собой всего лишь орудие их собственной профессиональной деятельности. Поэтому информационная система обязана обладать простым, удобным, легко осваиваемым интерфейсом, который должен предоставить конечному пользователю все необходимые для его работы функции, но в то же время не дать ему возможность выполнять какие-либо лишние действия. Иногда этот интерфейс может быть графическим с меню, кнопками, подсказками и т.д. Сейчас очень популярны графические интерфейсы, и как мы увидим в далее, многие современные средства разработки информационных приложений прежде всего ориентированы на разработку графических интерфейсов.

И снова заметим, что вычислительные программные системы не обязательно обладают развитыми интерфейсами. Конечно, это зависит от степени отчуждаемости программного продукта. Если система предназначена для продажи, то она должна обладать хорошим интерфейсом хотя бы в целях маркетинга. Но как правило, серьезные вычислительные программы почти уникальны.

Задачи информационных систем

Конкретные задачи, которые должны решаться информационной системой, зависят от той прикладной области, для которой предназначена система. Области применения информационных приложений разнообразны: банковское дело,

страхование, медицина, транспорт, образование и т.д. Трудно найти область деловой активности, в которой сегодня можно было обойтись без использования информационных систем. ***Можно выделить некоторое количество задач, не*** зависящих от специфики прикладной области. Естественно, такие задачи связаны с общими чертами информационных систем, рассмотренных в предыдущем разделе. Прежде всего, наиболее существенной составляющей является хранение информации, которая долго накапливается и утрата которой невосполнима.

В качестве примера рассмотрим ситуацию, существующую в Зеленчукской астрофизической лаборатории. В этой лаборатории в горах в районе Нижнего Архыза установлен один из крупнейших в мире зеркальных телескопов (диаметр зеркала - 6 метров). Уникальные природные условия этого района Северного Кавказа позволяют максимально эффективно использовать возможности обсерватории. В самом Зеленчуке имеется крупнейший в России радиотелескоп. Комбинированное использование этих ресурсов в течение многих лет (более 10) позволило астрофизикам накопить уникальную информацию относительно разного рода космических объектов. К сожалению, компьютерные возможности лаборатории в первые годы ее существования были весьма ограничены, и поэтому накапливаемые данные хранились в основном на магнитных лентах. Известно, что любой магнитный носитель стареет, а магнитные ленты еще и пересыхают. В результате основной проблемой группы поддержки информационных ресурсов уже несколько лет является копирование старых магнитных лент на новые носители. Старые ленты часто не читаются, и приходится тратить громадные усилия и средства для их реанимирования. Здесь уже не до создания информационной системы. Хотя, конечно, астрофизикам очень нужны информационные системы, позволяющие хотя бы частично автоматизировать огромные объемы работ по анализу и обобщению накопленной информации.

Основной вывод, который можно сделать на основе этой истории, состоит в том, что если некоторая организация планирует долговременное накопление ценной информации, то с самого начала должны быть обдуманы надежные способы ее долговременного хранения. В частности, информация, накопленная Зеленчукской лабораторий, должна храниться вечно.

Следующей задачей, которую должно выполнять большинство информационных систем, - это ***хранение данных, обладающих разными структурами.*** Трудно представить себе более или менее развитую информационную систему, которая работает с одним однородным файлом данных. Более того, разумным требованием к информационной системе является то, чтобы она могла развиваться. Могут появиться новые функции, для выполнения которых требуются дополнительные данные с новой структурой. При этом вся накопленная ранее информация должна остаться сохранной.

Еще один ***класс задач относится к обеспечению удобного и соответствующего целям информационной системы пользовательского интерфейса.*** Более или менее просто выяснить функциональные компоненты интерфейса, например, какого вида должны предлагаться формы и какого вида

должны выдаваться отчеты. Но построение действительно удобного и неутомительного для пользователя интерфейса - это задача дизайнера интерфейса. Простой аналог: при наличии полного набора качественной мебели хороший дизайнер сможет красиво оформить удобную для жизни квартиру (все на месте и под рукой). Плохой же дизайнер, скорее всего, добьется лишь того, что сможет запихнуть в квартиру всю мебель, а потом хозяину квартиры все время будет казаться, что у него слишком много ненужной мебели и ничего невозможно найти.

На первый взгляд упомянутая задача кажется не очень существенной. Можно полагать, что если информационная система обеспечивает полный набор функций и ее интерфейс обеспечивает доступ к любой из этих функций, то конечные пользователи должны быть удовлетворены. На самом деле, это не так. Пользователи часто судят о качестве системы в целом исходя из качества ее интерфейса. Более того, эффективность использования системы зависит от качества интерфейса.

Таким образом можно сказать, что компьютер - это машина, а не биологическое существо. Как любое техническое устройство***, компьютер обменивается информацией с человеком посредством набора определенных правил, обязательных как для машины, так и для человека. Эти правила в компьютерной литературе называются интерфейсом***.

Интерфейс может быть понятным и непонятным, дружественным и нет. К нему подходят многие прилагательные.

Интерфейс, по определению - это правила взаимодействия операционной системы с пользователями, а также соседними уровнями в сети ЭВМ. От интерфейса зависит технология общения человека с компьютером.

Классификация интерфейсов

То есть интерфейс - это набор правил. Как любые правила, их можно обобщить, собрать в "кодекс", сгруппировать по общему признаку. Таким образом, мы пришли к понятию "***вид интерфейса" как объединение по схожести способов взаимодействия человека и компьютеров.*** Можно предложить следующую схематическую классификацию различных интерфейсов общения человека и компьютера.

Современными видами интерфейсов являются:

1) *Командный интерфейс.* Командный интерфейс называется так по тому, что в этом виде интерфейса человек подает "команды" компьютеру, а компьютер их выполняет и выдает результат человеку. Командный интерфейс реализован в виде пакетной технологии и технологии командной строки.

2) *WIMP - интерфейс* (Window - окно, Image - образ, Menu - меню, Pointer - указатель). Характерной особенностью этого вида интерфейса является то, что диалог с пользователем ведется не с помощью команд, а с помощью графических образов - меню, окон, других элементов. Хотя и в этом интерфейсе подаются команды машине, но это делается "опосредственно", через графические образы. Этот вид интерфейса реализован на двух уровнях технологий: простой графический интерфейс и "чистый" WIMP - интерфейс.

3) *SILK - интерфейс* (Speech - речь, Image - образ, Language - язык, Knowlege - знание). Этот вид интерфейса наиболее приближен к обычной, человеческой форме общения. В рамках этого интерфейса идет обычный "разговор" человека и компьютера. При этом компьютер находит для себя команды, анализируя человеческую речь и находя в ней ключевые фразы. Результат выполнения команд он также преобразует в понятную человеку форму. Этот вид интерфейса наиболее требователен к аппаратным ресурсам компьютера, и поэтому его применяют в основном для военных целей.

1. Общественный интерфейс - основан на семантических сетях.

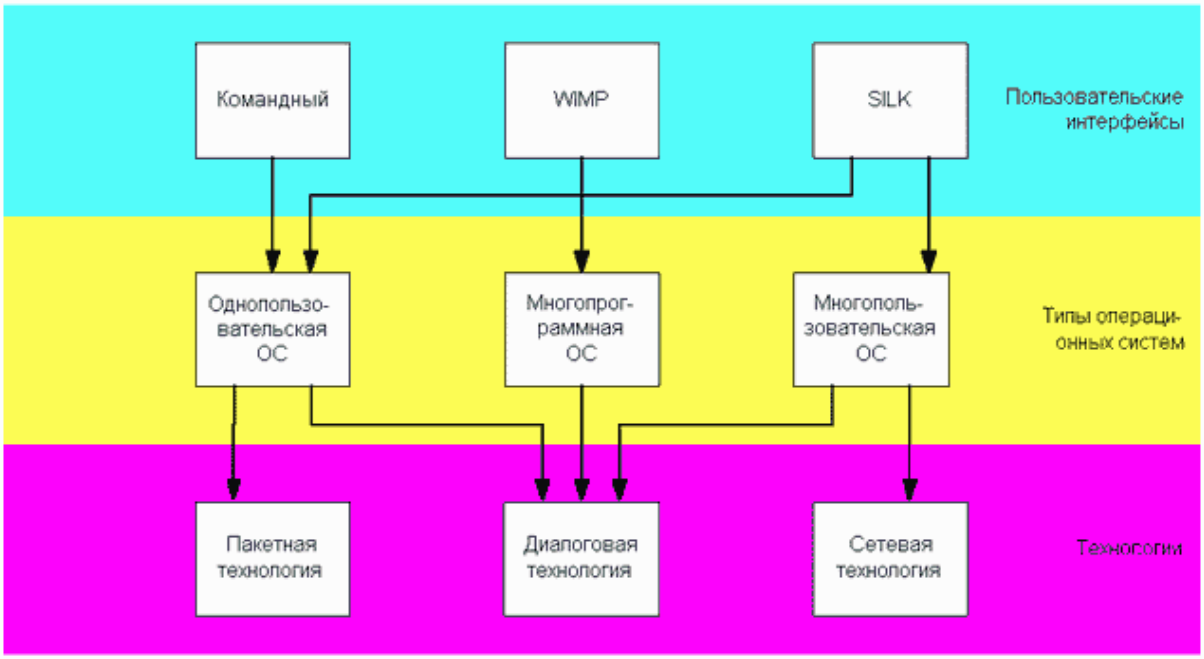


Рис. А.1. *Взаимодействие типов операционных систем, пользовательских интерфейсов и технологий их реализации.*

В следующих главах Вы подробнее познакомитесь с этими видами интерфейсов.

Пакетная технология

Исторически этот вид технологии появился первым. Она существовала уже на релейных машинах Зюса и Цюзе (Германия, 1937 год).

Идея ее проста: на вход компьютера подается последовательность символов, в которых по определенным правилам указывается последовательность запущенных на выполнение программ. После выполнения очередной программы запускается следующая и т.д. Машина по определенным правилам находит для себя команды и данные. В качестве этой последовательности может выступать, например, перфолента, стопка перфокарт, последовательность нажатия клавиш электрической пишущей машинки (типа CONSUL). Машина также выдает свои сообщения на перфоратор, алфавитно-цифровое печатающее устройство (АЦПУ), ленту пишущей машинки.

С появлением алфавитно-цифровых дисплеев началась эра по-настоящему пользовательской технологии - командной строки.

Технология командной строки.

При этой технологии в качестве единственного способа ввода информации от человека к компьютеру служит клавиатура, а компьютер выводит информацию человеку с помощью алфавитно-цифрового дисплея (монитора). Эту комбинацию (монитор + клавиатура) стали называть терминалом, или консолью.

Команды набираются в командной строке. Командная строка представляет собой символ приглашения и мигающий прямоугольник - При нажатии клавиши на месте курсора появляются символы, а сам курсор смещается вправо. Это очень похоже на набор команды на пишущей машинке. Однако, в отличие от нее, буквы отображаются на дисплее, а не на бумаге, и неправильно набранный символ можно стереть. Команда заканчивается нажатием клавиши Enter (или Return.) После этого осуществляется переход в начало следующей строки. Именно с этой позиции компьютер выдает на монитор результаты своей работы. Затем процесс повторяется.

Технология командной строки уже работала на монохромных алфавитно-цифровых дисплеях. Поскольку вводить позволялось только буквы, цифры и знаки препинания, то технические характеристики дисплея были не существенны. В качестве монитора можно было использовать телевизионный приемник и даже трубку осциллографа.

Преобладающим видом файлов при работе с командным интерфейсом стали текстовые файлы - их и только их можно было создать при помощи клавиатуры.

Графический интерфейс

Как и когда появился графический интерфейс?

Его идея зародилась в середине 70-х годов, когда в исследовательском центре Xerox Palo Alto Research Center (PARC) была разработана концепция визуального интерфейса. Предпосылкой графического интерфейса явилось уменьшение времени реакции компьютера на команду, увеличение объема оперативной памяти, а также развитие технической базы компьютеров. Аппаратным основанием концепции, конечно же, явилось появление алфавитно-цифровых дисплеев на компьютерах, причем на этих дисплеях уже имелись такие эффекты, как "мерцание" символов, инверсия цвета (смена начертания белых символов на черном фоне обратным, то есть черных символов на белом фоне), подчеркивание символов. Эти эффекты распространились не на весь экран, а только на один или более символов.

Следующим шагом явилось создание цветного дисплея, позволяющего выводить, вместе с этими эффектами, символы в 16 цветах на фоне с палитрой (то есть цветовым набором) из 8 цветов. После появления графических дисплеев, с возможностью вывода любых графических изображений в виде множества точек на экране различного цвета, фантазии в использовании экрана вообще не стало границ! Первая система с графическим интерфейсом 8010 Star Information System группы PARC, таким образом, появилась за четыре месяца до выхода в свет первого компьютера фирмы IBM в 1981 году. Первоначально визуальный интерфейс использовался только в программах. Постепенно он стал переходить и

на операционные системы, используемых сначала на компьютерах Atari и Apple Macintosh, а затем и на IBM -- совместимых компьютерах.

С более раннего времени, и под влиянием также и этих концепций, проходил процесс по унификации в использовании клавиатуры и мыши прикладными программами. Слияние этих двух тенденций и привело к созданию того пользовательского интерфейса, с помощью которого, при минимальных затратах времени и средств на переучивание персонала, можно работать с любыми программным продуктом. Описание этого интерфейса, общего для всех приложений и операционных систем, и посвящена данная часть.

Графический интерфейс пользователя за время своего развития прошел две стадии. Об эволюции графического интерфейса с 1974 по настоящее время будет рассказано ниже.

Простой графический интерфейс.

На первом этапе графический интерфейс очень походил на технологию командной строки. Отличия от технологии командной строки заключались в следующим.

a) При отображении символов допускалось выделение части символов цветом, инверсным изображением, подчеркиванием и мерцанием. Благодаря этому повысилась выразительность изображения.

b) В зависимости от конкретной реализации графического интерфейса курсор может представляться не только мерцающим прямоугольником, но и некоторой областью, охватывающей несколько символов и даже часть экрана. Эта выделенная область отличается от других, невыделенных частей (обычно цветом).

c) Нажатие клавиши Enter не всегда приводит к выполнению команды и переходу к следующей строке. Реакция на нажатие любой клавиши во многом зависит от того, в какой части экрана находился курсор.

d) Кроме клавиши Enter, на клавиатуре все чаще стали использоваться "серые" клавиши управления курсором (см. раздел, посвященный клавиатуре в выпуске 3 данной серии.)

e) Уже в этой редакции графического интерфейса стали использоваться манипуляторы (типа мыши, трекбола и т.п. - см. рисунок А.4.) Они позволяли быстро выделять нужную часть экрана и перемещать курсор.

Рис. А.4. *Манипуляторы*

Подводя итоги, можно привести следующие отличительные особенности этого интерфейса.

1) Выделение областей экрана.

2) Переопределение клавиш клавиатуры в зависимости от контекста.

3) Использование манипуляторов и серых клавиш клавиатуры для управления курсором.

4) Широкое использование цветных мониторов.

Появление этого типа интерфейса совпадает с широким распространением операционной системы MS-DOS. Именно она внедрила этот интерфейс в массы, благодаря чему 80-е годы прошли под знаком совершенствования этого типа интерфейса, улучшения характеристик отображения символов и других параметров монитора.

Типичным примером использования этого вида интерфейса является файловая оболочка Nortron Commander (о файловых оболочках смотри ниже) и текстовый редактор Multi-Edit. А текстовые редакторы Лексикон, ChiWriter и текстовый процессор Microsoft Word for Dos являются примером, как этот интерфейс превзошел сам себя.

WIMP - интерфейс

Вторым этапом в развитии графического интерфейса стал "чистый" интерфейс WIMP, Этот подвид интерфейса характеризуется следующими особенностями.

1. Вся работа с программами, файлами и документами происходит в окнах - определенных очерченных рамкой частях экрана.

2. Все программы, файлы, документы, устройства и другие объекты представляются в виде значков - иконок. При открытии иконки превращаются в окна.

3. Все действия с объектами осуществляются с помощью меню. Хотя меню появилось на первом этапе становления графического интерфейса, оно не имело в нем главенствующего значения, а служило лишь дополнением к командной строке. В чистом WIMP - интерфейсе меню становится основным элементом управления.

4. Широкое использование манипуляторов для указания на объекты. Манипулятор перестает быть просто игрушкой - дополнением к клавиатуре, а становится основным элементом управления. С помощью манипулятора УКАЗЫВАЮТ на любую область экрана, окна или иконки, ВЫДЕЛЯЮТ ее, а уже потом через меню или с использованием других технологий осуществляют управление ими.

Следует отметить, что WIMP требует для своей реализации цветной растровый дисплей с высоким разрешением и манипулятор. Также программы, ориентированные на этот вид интерфейса, предъявляют повышенные требования к производительности компьютера, объему его памяти, пропускной способности шины и т.п. Однако этот вид интерфейса наиболее прост в усвоении и интуитивно понятен. Поэтому сейчас WIMP - интерфейс стал стандартом де-факто.

Ярким примером программ с графическим интерфейсом является операционная система Microsoft Windows.

Речевая технология

С середины 90-х годов, после появления недорогих звуковых карт и широкого распространения технологий распознавания речи, появился так называемый "речевая технология" SILK - интерфейса. При этой технологии команды подаются голосом путем произнесения специальных зарезервированных слов - команд. Основными такими командами (по правилам системы "Горыныч") являются:

- "Проснись" - включение голосового интерфейса.
- "Отдыхай" - выключение речевого интерфейса.
- "Открыть" - переход в режим вызова той или иной программы. Имя программы называется в следующем слове.
- "Буду диктовать" - переход из режима команд в режим набора текста голосом.
- "Режим команд" - возврат в режим подачи команд голосом.
- и некоторые другие.

Слова должны выговариваться четко, в одном темпе. Между словами обязательна пауза. Из-за неразвитости алгоритма распознавания речи такие системы требует индивидуальной предварительной настройки на каждого конкретного пользователя.

"Речевая" технология является простейшей реализацией SILK - интерфейса.

Биометрическая технология ("Мимический интерфейс".)

Эта технология возникла в конце 90-х годов XX века. Для управления компьютером используется выражение лица человека, направление его взгляда, размер зрачка и другие признаки. Для идентификации пользователя используется рисунок радужной оболочки его глаз, отпечатки пальцев и другая уникальная информация. Изображения считываются с цифровой видеокамеры, а затем с помощью специальных программ распознавания образов из этого изображения выделяются команды. Эта технология, по-видимому, займет свое место в программных продуктах и приложениях, где важно точно идентифицировать пользователя компьютера.

Семантический (Общественный) интерфейс.

Этот вид интерфейса возник в конце 70-х годов XX века, с развитием искусственного интеллекта. Его трудно назвать самостоятельным видом интерфейса - он включает в себя и интерфейс командной строки, и графический, и речевой, и мимический интерфейс. Основная его отличительная черта - это отсутствие команд при общении с компьютером. Запрос формируется на естественном языке, в виде связанного текста и образов. По своей сути это трудно называть интерфейсом - это уже моделирование "общения" человека с компьютером.

С середины 90-х годов XX века автор уже не встречал публикаций, относящихся к семантическому интерфейсу. Похоже, что в связи с важным военным значением этих разработок (например, для автономного ведения современного боя машинами - роботами, для "семантической" криптографии) эти направления были засекречены. Информация, что эти исследования

продолжаются, иногда появляется в периодической печати (обычно в разделах компьютерных новостей).

Тема 2. Типы пользовательских интерфейсов и этапы их разработки.

На ранних этапах развития вычислительной техники пользовательский интерфейс рассматривался как средство общения человека с операционной системой и был достаточно примитивным. В основном он позволял запустить задание на выполнение, связать с ним конкретные данные и выполнить некоторые процедуры обслуживания вычислительной установки.

Со временем по мере совершенствования аппаратных средств появилась возможность создания интерактивного программного обеспечения, использующего специальные пользовательские интерфейсы. В настоящее время основной проблемой является разработка интерактивных интерфейсов к сложным программным продуктам, рассчитанным на использование непрофессиональными пользователями. В последние годы были сформулированы основные концепции построения таких пользовательских интерфейсов и предложено несколько методик их создания.

Пользовательский интерфейс представляет собой совокупность программных и аппаратных средств, обеспечивающих взаимодействие пользователя с компьютером. Основу такого взаимодействия составляют диалоги. Под диалогом в данном случае понимают регламентированный обмен информацией между человеком и компьютером, осуществляемый в реальном масштабе времени и направленный на совместное решение конкретной задачи: обмен информацией и координация действий. Каждый диалог состоит из отдельных процессов ввода-вывода, которые физически обеспечивают связь пользователя и компьютера.

Обмен информацией осуществляется передачей сообщений и управляющих сигналов. Сообщение - порция информации, участвующая в диалоговом обмене. Различают:

•входные сообщения, которые генерируются человеком с помощью средств ввода: клавиатуры, манипуляторов, например мыши и т. п.;

•выходные сообщения, которые генерируются компьютером в виде текстов, звуковых сигналов и/или изображений и выводятся пользователю на экран монитора или другие устройства вывода информации (рис.1).

Рис.1. Организация взаимодействия компьютера и пользователя

В основном пользователь генерирует сообщения следующих типов: запрос информации, запрос помощи, запрос операции или функции, ввод или изменение информации, выбор поля кадра и т. д. В ответ он получает: подсказки или справки, информационные сообщения, не требующие ответа, приказы, требующие действий, сообщения об ошибках, нуждающиеся в ответных действиях, изменение формата кадра и т. д.

Ниже перечислены основные устройства, обеспечивающие выполнение операций ввода-вывода.

Для вывода сообщений:

- монохромные и цветные мониторы - вывод оперативной текстовой и графической информации;
- принтеры - получение «твердой копии» текстовой и графической информации;
- графопостроители - получение твердой копии графической информации;
- синтезаторы речи - речевой вывод;
- звукогенераторы - вывод музыки и т. п.

Для ввода сообщений:

- клавиатура - текстовый ввод;
- планшеты - графический ввод;
- сканеры - графический ввод;
- манипуляторы, световое перо, сенсорный экран - позиционирование и выбор информации на экране и т. п.

Типы интерфейсов.

По аналогии с процедурным и объектным подходом к программированию различают процедурно-ориентированный и объектно-ориентированный подходы к разработке интерфейсов (рис.2).

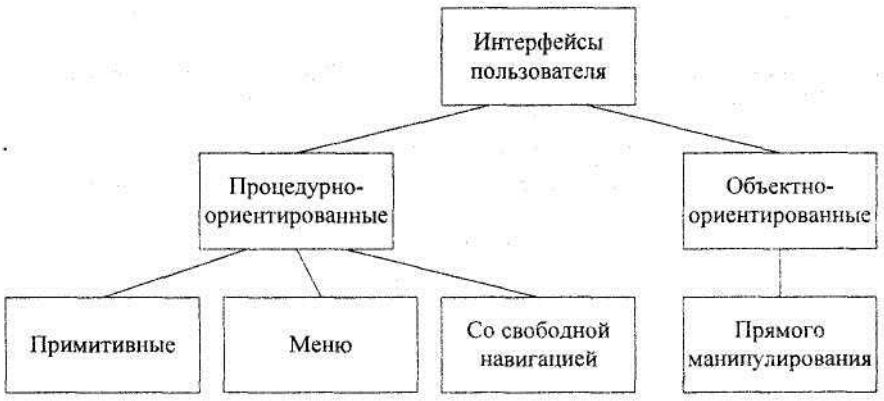


Рис. 2. Типы интерфейсов

Процедурно-ориентированные интерфейсы используют традиционную модель взаимодействия с пользователем, основанную на понятиях «процедура» и «операция». В рамках этой модели программное обеспечение предоставляет пользователю возможность выполнения некоторых действий, для которых

пользователь определяет соответствующие данные и следствием выполнения которых является получение желаемых результатов.

Объектно-ориентированные интерфейсы используют несколько иную модель взаимодействия с пользователем, ориентированную на манипулирование объектами предметной области. В рамках этой модели пользователю предоставляется возможность напрямую взаимодействовать с каждым объектом и инициировать выполнение операций, в процессе которых взаимодействуют несколько объектов. Задача пользователя формулируется как целенаправленное изменение некоторого объекта, имеющего внутреннюю структуру, определенное содержание и внешнее символьное или графическое представление. Объект при этом понимается в широком смысле слова, например, модель реальной системы или процесса, база данных, текст и т, п. Пользователю предоставляется возможность создавать объекты, изменять их параметры и связи с другими объектами, а также инициировать взаимодействие этих объектов. Элементы интерфейсов данного типа включены в пользовательский интерфейс Windows, например, пользователь может «взять» файл и «переместить» его в другую папку. Таким образом, он инициирует выполнение операции перемещения файла.

Применение процедурно-ориентированных интерфейсов в данном случае не означает использования структурного подхода к разработке соответствующего программного обеспечения. Более того, реализация современного процедурно-ориентированного пользовательского интерфейса на базе структурного подхода является очень сложной и трудоемкой задачей.

Таблица 1.

Процедурно-ориентированные пользовательские интерфейсы	Объектно-ориентированные пользовательские интерфейсы
Обеспечивают пользователей функциями, необходимыми для выполнения задач	Обеспечивают пользователям возможность взаимодействия с объектами
Акцент делается на задачи	Акцент делается на входные данные и результаты
Пиктограммы представляют приложения, окна или операции	Пиктограммы представляют объекты
Содержание папок и справочников отображается с помощью таблиц и списков	Папки и справочники являются визуальными контейнерами объектов

В табл.1 перечислены основные отличия пользовательских моделей интерфейсов процедурного и объектно-ориентированного типов.

Различают ***процедурно-ориентированные*** интерфейсы трех типов: «примитивные», меню и со свободной навигацией.

Примитивным называют интерфейс, который организует взаимодействие с пользователем в консольном режиме. Обычно такой интерфейс реализует конкретный сценарий работы программного обеспечения, например: ввод данных - решение задачи - вывод результата (рис.3, а). Единственное отклонение от последовательного процесса, которое обеспечивается данным интерфейсом, заключается в организации цикла для обработки нескольких наборов данных (рис.

3, б). Подобные интерфейсы в настоящее время используют только в процессе обучения программированию или в тех случаях, когда вся программа реализует одну функцию, например, в некоторых системных утилитах.

Интерфейс-меню в отличие от примитивного интерфейса позволяет пользователю выбирать необходимые операции из специального списка, выводимого ему программой. Эти интерфейсы предполагают реализацию множества сценариев работы, последовательность действий в которых определяется пользователем.

Различают одноуровневые и иерархические меню. Первые используют для сравнительно простого управления вычислительным процессом, когда вариантов немного (не более 5-7), и они включают операции одного типа, например, Создать, Открыть, Закрыть и т. п. Вторые - при большом количестве вариантов или их очевидных различиях, например, операции с файлами и операции с данными, хранящимися в этих файлах.

Интерфейсы данного типа несложно реализовать в рамках структурного подхода к программированию. На рис. 4 показана типичная структура алгоритма программы, организующей одноуровневое меню. Алгоритм программы с многоуровневым меню обычно строится по уровням, причем выбор команды на каждом уровне осуществляется так же, как для одноуровневого меню.

Интерфейс-меню предполагает, что программа находится либо в состоянии Уровень меню, либо в состоянии Выполнение операции. В состоянии Уровень меню осуществляется вывод меню соответствующего уровня и выбор нужного пункта меню, а в состоянии Выполнение операции реализуется сценарий выбранной операции. В порядке исключения иногда пользователю предоставляется возможность завершения операции независимо от стадии выполнения сценария и/или программы, например, по нажатию клавиши Esc.

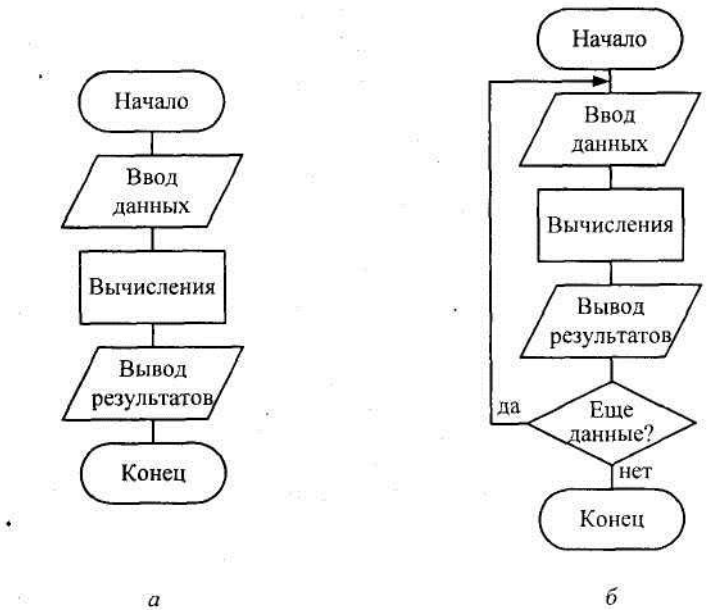


Рис.3. Типичная структура алгоритма программ с примитивным интерфейсом: *а* - последовательный; *б* - с возможностью повторения

Древовидная организация меню предполагает строго ограниченную навигацию: либо переходы «вверх» к корню дерева, либо - «вниз» по выбранной ветви. Каждому уровню иерархического меню соответствует свое определенное окно, содержащее пункты данного уровня. При этом возможны два варианта реализации меню: каждое окно меню занимает весь экран или на экране одновременно присутствуют несколько меню разных уровней. Во втором случае окна меню появляются при выборе пунктов соответствующего верхнего уровня — «выпадающие» меню.

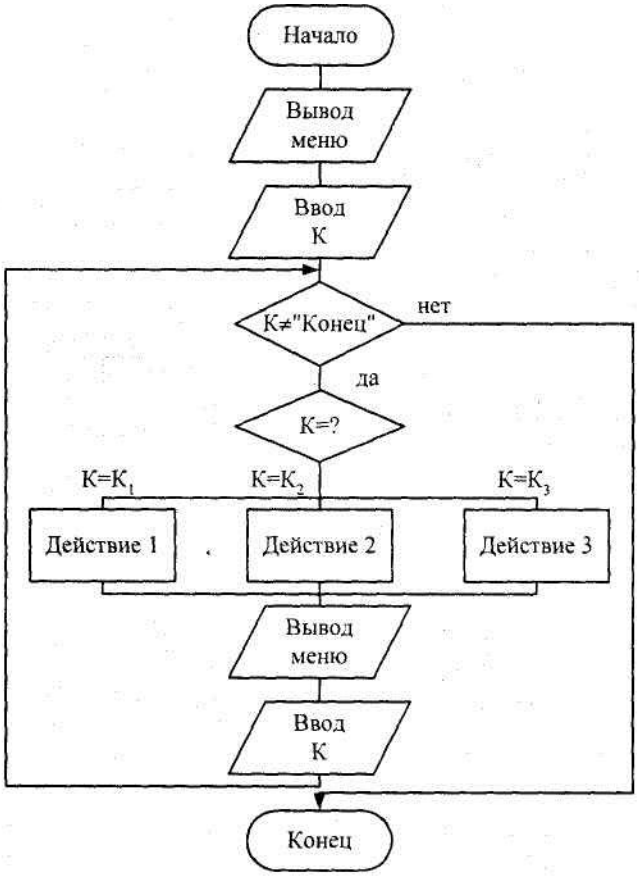


Рис. 8.4. Типичная структура алгоритма программы с одноуровневым меню

В условиях ограниченной навигации независимо от варианта реализации поиск требуемого пункта более чем двухуровневого меню может оказаться непростой задачей.

Интерфейсы-меню в настоящее время также используют редко и только для сравнительно простого программного обеспечения или в разработках, которые должны быть выполнены по структурной технологии и без использования специальных библиотек.

Интерфейсы со свободной навигацией также называют графическими пользовательскими интерфейсами (GUI - Graphic User Interface) или интерфейсами WYSIWYG (What You See Is What You Get - что видишь, то и получишь, т. е., что пользователь видит на экране, то он и получит при печати). Эти названия подчеркивают, что интерфейсы данного типа ориентированы на использование экрана в графическом режиме с высокой разрешающей способностью.

Графические интерфейсы поддерживают концепцию интерактивного взаимодействия с программным обеспечением, осуществляя визуальную

обратную связь с пользователем и возможность прямого манипулирования объектами и информацией на экране. Кроме того, интерфейсы данного типа поддерживают концепцию совместимости программ, позволяя перемещать между ними информацию (технология OLE, см. § 1.1).

В отличие от интерфейса-меню интерфейс со свободной навигацией обеспечивает возможность осуществления любых допустимых в конкретном состоянии операций, доступ к которым возможен через различные интерфейсные компоненты. Например, окна программ, реализующих интерфейс Windows, обычно содержат:

• меню различных типов: ниспадающее, кнопочное, контекстное;

• разного рода компоненты ввода данных.

Причем выбор следующей операции в меню осуществляется как мышью, так и с помощью клавиатуры.

Существенной особенностью интерфейсов данного типа является способность изменяться в процессе взаимодействия с пользователем, предлагая выбор только тех операций, которые имеют смысл в конкретной ситуации. Реализуют интерфейсы со свободной навигацией, используя событийное программирование и объектно-ориентированные библиотеки, что предполагает применение визуальных сред разработки программного обеспечения.

Объектно-ориентированные интерфейсы пока представлены только интерфейсом прямого манипулирования. Этот тип интерфейса предполагает, что взаимодействие пользователя с программным обеспечением осуществляется посредством выбора и перемещения пиктограмм, соответствующих объектам предметной области. Для реализации таких интерфейсов также используют событийное программирование и объектно-ориентированные библиотеки.

Сравним четыре указанных типа интерфейсов на конкретном несложном примере.

Пример 8.1. Разработать пользовательский интерфейс программы построения графиков или вывода таблицы функций, техническое задание на которую представлено в § 3.5.

Можно предложить четыре варианта интерфейса, соответствующие рассмотренным выше типам.

Вариант 1. Использование примитивного интерфейса предполагает, что пользователь сразу определяет все параметры, необходимые программе для построения графика или вывода таблицы, вводя их в ответ на соответствующие запросы программы, после чего программа выполняет необходимые вычисления и выводит результат. Если допустить, что программа будет запрашивать подтверждения завершения обработки, то процесс построения графиков/таблиц можно зациклить. В зависимости от используемых средств мы получим сравнительно простую программу, удовлетворяющую функциональным спецификациям, но ориентированную на единственный сценарий: ввод - обработка - вывод (см. рис. 8.3, б). Данный вариант не удобен для пользователя.

Вариант 2. Можно использовать одноуровневое меню, которое будет включать команды: Функция, Отрезок, Шаг, Тип результата, Выполнить и Выход. При выборе первого пункта меню определяется функция, второго -интервал, третьего - шаг, четвертого - тип результата, пятого - осуществляется операция, и,

наконец, последний пункт обеспечивает возможность выхода из программы (рис. 8.5).

Очевидно, что в этом случае обеспечивается более гибкое управление для пользователя, так как фактически предусмотрены следующие сценарии работы:

Ввод функции - Ввод отрезка - Ввод шага - Уточнение вида результата: график/таблица - Вывод результата;

Изменение отрезка - Вывод результата;

Изменение шага - Вывод результата;

Изменение вида результата: график/таблица - Вывод результата и др.

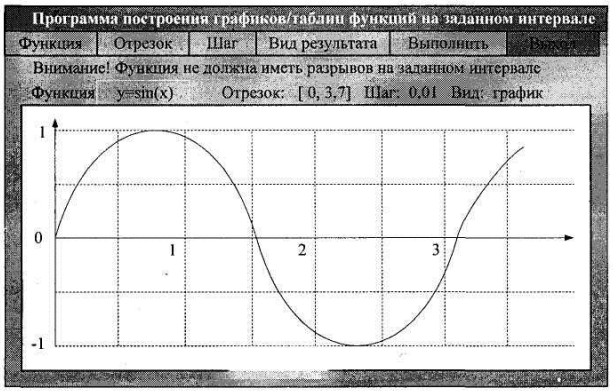


Рис.8.5. Внешний вид окна программы

Вариант 3. Интерфейс со свободной навигацией для данной программы представлен на рис. 8.6. График строится по нажатии кнопки Построить.

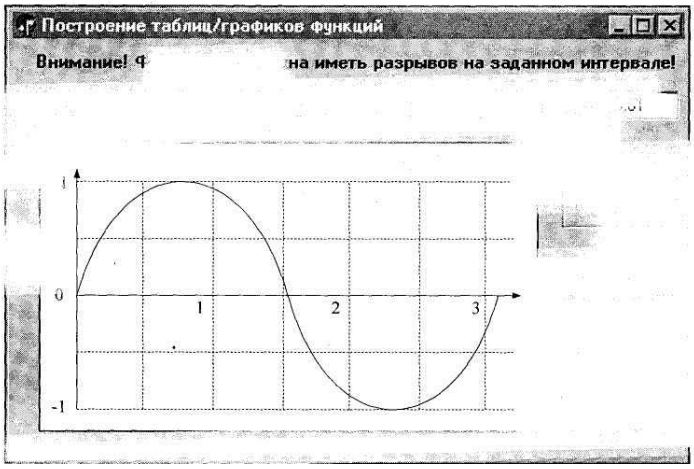
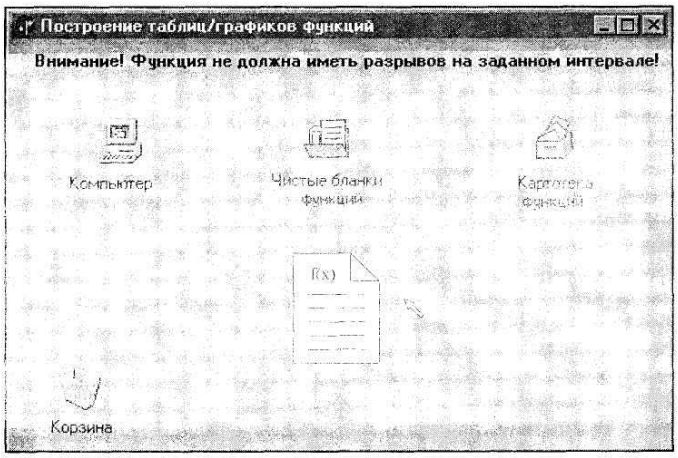


Рис. 8.6. Внешний вид окна программы построения графиков/таблиц функций (интерфейс со свободной навигацией) (естественно, обработчик этого события должен предусматривать анализ данных на полноту и совместимость). Менять данные можно в любой момент и в любом порядке, используя соответствующие компоненты ввода-вывода.

Вариант 4. Интерфейс прямого манипулирования для данной программы представлен на рис. 8.7. Для того чтобы ввести новую формулу, необходимо взять чистый бланк из папки. Бланк раскрывается двойным щелчком мыши, после чего его необходимо заполнить. Затем его можно «обсчитать», перенеся на пиктограмму компьютера. Заполненные бланки, которые могут еще понадобиться, «кладутся» в папку Функции, остальные - в «корзину».

Менять данные и тип результатов можно в любой момент и в любом порядке, «раскрыв» бланк.



Как уже упоминалось в § 3.5, различают также однодокументные (SDI -Single Document Interface) и многодокументные (MDI - Multiple Document Interface) интерфейсы. Однодокументные или «однооконные» интерфейсы организуют работу, как следует из названия, только с одним

Тема 3. Психофизические особенности человека, связанные с восприятием, запоминанием и обработкой информации

При проектировании пользовательских интерфейсов необходимо учитывать психофизические особенности человека, связанные с восприятием, запоминанием и обработкой информации.

Исследованием принципов работы мозга человека занимается когнитивная психология. Специалисты в этой области предлагают упрощенную информационно-процессуальную модель мозга, представленную на рис. 8.8.

Информация о внешнем мире поступает в наш мозг в огромных количествах. Часть мозга, которую условно можно назвать «процессором восприятия», постоянно без участия сознания перерабатывает ее, сравнивая с прошлым опытом, и помещает в хранилище уже в виде зрительных, звуковых и прочих образов. Любые внезапные или просто значимые для нас изменения в окружении привлекают наше внимание, и тогда интересующая нас информация поступает в кратковременную память. Если же наше внимание не было привлечено, то информация в хранилище пропадает, замещаясь следующими порциями.

Рис. 8.8. Упрощенная информационно-процессуальная модель мозга

В каждый момент времени фокус внимания может фиксироваться в одной точке. Поэтому, если возникает необходимость «одновременно» отслеживать несколько ситуаций, то обычно фокус перемещается с одного отслеживаемого элемента на другой. При этом внимание «рассредоточивается», и какие-то детали могут быть упущены. Например, при «прокрутке» текста или рисунка с использованием линейки прокрутки окна Windows приходится одновременно смотреть на текст, чтобы определить, где остановиться, и на ползунок. Поскольку текст важнее, фокус внимания перестает перемещаться на мышь, и она «соскакивает» с ползунка линейки. Следует иметь в виду, что обработка процессором восприятия требует некоторого времени и, если сигнал выдается в течение времени, меньшем времени обработки, то наш мозг его не воспринимает.

Существенно и то, что восприятие во многом основано на мотивации. Например, если человек голоден, то он в первую очередь будет замечать все съедобное, а если устал - то, войдя в комнату, он в первую очередь увидит диван или кровать.

Необходимо также учитывать, что в процессе переработки информации мозг сравнивает поступающие данные с предыдущими. Так, если показать человеку последовательность символов:

A, D, C,

то он может принять 13 за B.

При смене кадра мозг на некоторое время блокируется: он «осваивает» новую картинку, выделяя наиболее существенные детали. А значит, если необходима быстрая реакция пользователя, то резко менять картинку не стоит.

Краткосрочная память - самое «узкое» место «системы обработки информации» человека. Ее емкость приблизительно равна 7 ± 2 несвязанных объектов. Краткосрочная память является своего рода оперативной памятью мозга, именно с ней работает процессор познания, но не востребованная информация хранится в ней не более 30 с. Чтобы не забыть какую-нибудь важную для нас информацию, мы обычно повторяем ее «про себя», «обновляя» информацию в краткосрочной памяти. Таким образом, при проектировании интерфейсов следует иметь в виду, что подавляющему большинству людей

сложно, например, запомнить и ввести на другом экране число, содержащее более 5 цифр (7 - 2), или некоторое сочетание букв.

Люди вносят в каждую деятельность свое понимание того, как она должна выполняться. Это понимание -модель деятельности - базируется на прошлом опыте человека. Множество таких моделей хранится в долговременной памяти человека.

В долговременную память записываются постоянно повторяемые сведения или информация, связанная с сильными эмоциями. Долговременная память человека - хранилище информации с неограниченной емкостью и временем хранения. Однако доступ к этой информации весьма непрост: по всей вероятности, механизмы извлечения информации из памяти имеют ассоциативный характер. Специальная методика запоминания информации (мнемоника) использует именно это свойство памяти: для запоминания информации ее «привязывают» к тем данным, которые память уже хранит и позволяет легко получить.

Поскольку доступ к долговременной памяти затруднен, целесообразно рассчитывать не на то, что пользователь вспомнит нужную информацию, а на то, что он ее узнает. Именно поэтому интерфейс типа меню так широко используется.

Особенности восприятия цвета. Цвет в сознании человека ассоциируется с эмоциональным фоном. Известно, что теплые цвета: красный, оранжевый, желтый человека возбуждают, а холодные: синий, фиолетовый, серый - успокаивают. Причем цвет для человека является очень сильным раздражителем, поэтому применять цвета в интерфейсе необходимо крайне осторожно.

Следует иметь в виду, что обилие оттенков привлекает внимание, но быстро утомляет. Поэтому не стоит ярко раскрашивать окна, с которыми пользователь будет долго работать. Необходимо учитывать и индивидуальные особенности восприятия цветов человеком, например, примерно каждый десятый человек плохо различает какие-либо цвета, поэтому в ответственных случаях необходимо предоставить пользователю возможность настройки цветов.

Особенности восприятия звука. В интерфейсах звук обычно используют с разными целями: для привлечения внимания, как фон, обеспечивающий некоторое состояние пользователя, как источник дополнительной информации и т. п. Применяя звук, следует учитывать, что большинство людей очень чувствительны к звуковым сигналам, особенно, если последние указывают на наличие ошибки. Поэтому при создании звукового сопровождения целесообразно предусматривать возможность его отключения.

Субъективное восприятие времени. Человеку свойственно субъективное восприятие времени. Считают, что внутреннее время связано со скоростью и количеством воспринимаемой и обрабатываемой информации. Занятый человек обычно времени не замечает. Зато в состоянии ожидания время тянется бесконечно, что связано с тем, что в это время мозг оказывается в состоянии информационного вакуума. (К аналогичному состоянию приводит и усталость: информация поступает, но больше обрабатывается, а потому и ход времени замедляется.)

Доказано, что при ожидании более 1-2 с пользователь может отвлечься, «потерять мысль», что неблагоприятно сказывается на результатах работы и

увеличивает усталость, так как каждый раз после ожидания много сил тратится на включение в работу.

Сократить время ожидания можно, заняв пользователя, но не отвлекая его от работы. Например, можно предоставить ему какую-либо информацию для обдумывания. По возможности целесообразно выводить пользователю промежуточные результаты: во-первых, он будет занят их обдумыванием, во-вторых, по ним он сможет оценить будущие результаты и отменит операцию, если они его не удовлетворяют.

Известны попытки использования для «развлечения» пользователя анимации, например, в Windows при копировании файлов демонстрируется «ролик» с летающими листочками. Однако следует иметь в виду, что, когда какую-либо анимацию смотришь первый раз, то это интересно, а когда в течение получаса наблюдаешь, как «летают» листочки при получении информации из Интернета, то это начинает раздражать.

Чтобы уменьшить раздражение, возникающее при ожидании, необходимо соблюдать основное правило: информировать пользователя, что заказанные им операции потребуют некоторого времени выполнения. Обычно для этого используют индикаторы оставшегося времени, анимированные объекты, как в Интернете, и изменение формы курсора мыши на песочные часы. Очень важно точно обозначить момент, когда система готова продолжать работу. Обычно для этого используют значительные изменения внешнего вида экрана.

В конечном итоге взаимодействие пользователя с интерфейсом будет определяться не только физическими возможностями и особенностями человека по восприятию, обработке и запоминанию информации, представленной в различных формах, а также по выполнению им разнообразных действий, но и пользовательской моделью интерфейса.

Пользовательская и программная модели интерфейса

Существуют три совершенно различные модели пользовательского интерфейса: модель программиста, модель пользователя и программная модель. Программист, разрабатывая пользовательский интерфейс, исходит из того, управление какими операциями ему необходимо реализовать в пользовательском интерфейсе, и как это осуществить, не затрачивая ни существенных ресурсов компьютера, ни своих сил и времени. Его интересуют функциональность, эффективность, технологичность, внутренняя стройность и другие не связанные с удобством пользователя характеристики программного обеспечения. Именно поэтому большинство интерфейсов существующих программ вызывают серьезные нарекания пользователей.

С точки зрения здравого смысла хорошим следует считать интерфейс, при работе с которым пользователь получает именно то, что он ожидал. Представление пользователя о функциях интерфейса можно описать в виде пользовательской модели интерфейса.

Пользовательская модель интерфейса - это совокупность обобщенных представлений конкретного пользователя или некоторой группы пользователей о процессах, происходящих во время работы программы или программной системы.

Эта модель базируется на особенностях опыта конкретных пользователей, который характеризуется:

• уровнем подготовки в предметной области разрабатываемого про граммного обеспечения;

• интуитивными моделями выполнения операций в этой предметной об ласти;

• уровнем подготовки в области владения компьютером;

• устоявшимися стереотипами работы с компьютером.

Для построения пользовательской модели необходимо изучить пере-численные выше особенности опыта предполагаемых пользователей про-граммного обеспечения. С этой целью используют опросы, тесты и даже фиксируют последовательность действий, осуществляемых в процессе вы-полнения некоторых операций, на пленку.

Приведение в соответствие моделей пользователя и программиста, а также построение на их базе программной модели (рис. 8.9) интерфейса задача не тривиальная. Причем, чем сложнее автоматизируемая предметная область, тем сложнее оказывается построить программную модель интерфейса, учитывающую особенности пользовательской модели и не требующую слишком больших затрат как в процессе разработки, так и во время работы. С этой точки зрения объектные интерфейсы кажутся наиболее перспективными, так как в их основе лежит именно отображение объектов предметной области, которыми оперируют пользователи. Хотя на настоящий момент времени их реализация достаточно трудоемка.

При создании программной модели интерфейса также следует иметь в виду, что изменить пользовательскую модель непросто. Повышение профес-сионального уровня пользователей и их подготовки в области владения компьютером в компетенцию разработчиков программного обеспечения не вхо-дит, хотя часто грамотно построенный интерфейс, который адекватно отображает сущность происходящих процессов, способствует росту квалификации пользователей.

Интуитивные модели выполнения операций в предметной области должны стать основой для разработки интерфейса, а потому в большинстве случаев их необходимо не менять, а уточнять и совершенствовать. Именно нежелание или невозможность следования интуитивным моделям выполнения операций приводит к созданию искусственных надуманных интерфейсов, которые негативно воспринимаются пользователями.

Иногда кажется, что единственно доступный для изменения элемент - устоявшийся стереотип работы с компьютером. Однако ломка стереотипов - процедура болезненная. На это стоит решаться, если некоторое революционное изменение значительно расширяет возможности пользователя или облегчает его работу, например, переход к Windows-интерфейсам существенно упростил работу

с компьютером огромному числу пользователей -

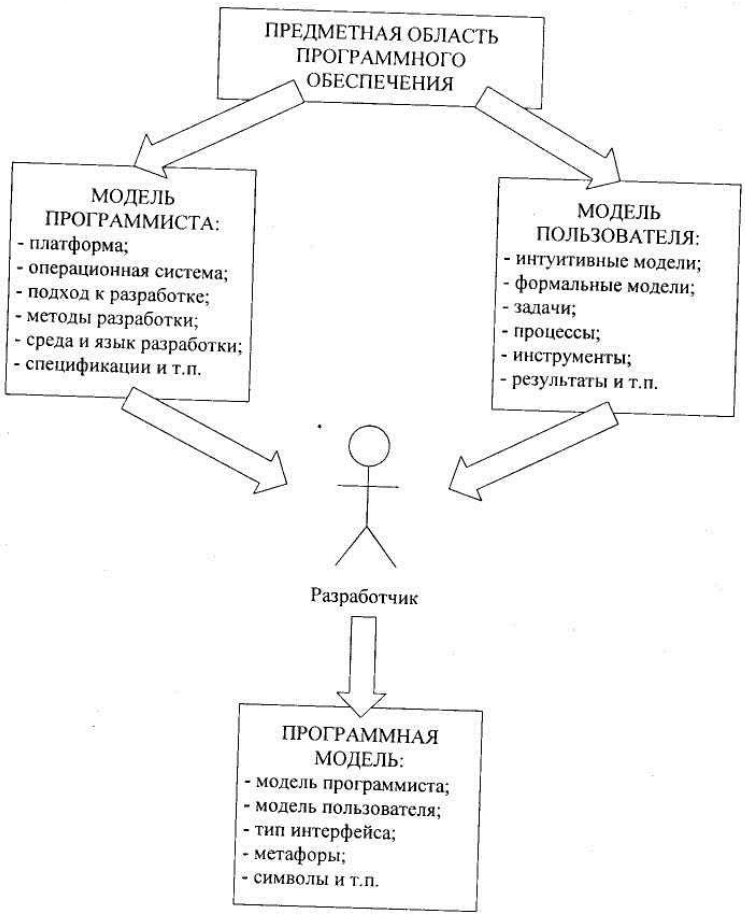


Рис. 8.9. Процесс проектирования пользовательского интерфейса

непрофессионалов. Ломая же стереотипы по мелочам или неточно следуя принятой концепции, разработчик рискует оттолкнуть пользователей, которые просто не будут понимать, что происходит. В качестве примера можно вспомнить хотя бы путаницу с вызовом программ двойным щелчком правой клавиши мыши по пиктограмме рабочем столе или одинарным, если пиктограммы вынесена на панель Quick Launch (Быстрый Доступ) Windows.

Критерии оценки интерфейса пользователем. Многочисленные опросы и обследования, проводимые ведущими фирмами по разработке программного обеспечения, показали, что основными критериями оценки интерфейсов пользователем являются:

• простота освоения и запоминания операций системы - конкретно оце нивают время освоения и продолжительность сохранения информации в па мяти;

• скорость достижения результатов при использовании системы - опре деляется количеством вводимых или выбираемых мышью команд и на строек;

• субъективная удовлетворенность при эксплуатации системы (удобство работы, утомляемость и т. д.).

Причем для пользователей-профессионалов, постоянно работающих с одним и тем же пакетом, на первое место достаточно быстро выходят второй и третий критерии, а для пользователей-непрофессионалов, работающих с программным

обеспечением периодически и выполняющих сравнительно несложные задачи - первый и третий.

С этой точки зрения на сегодняшний день наилучшими характеристиками для пользователей-профессионалов обладают интерфейсы со свободной навигацией, а для пользователей-непрофессионалов - интерфейсы прямого манипулирования. Давно замечено, что при выполнении операции копирования файлов при прочих равных условиях большинство профессионалов используют оболочки типа Far, а непрофессионалы - «перетаскивание объектов» Windows.

Классификации диалогов и общие принципы их разработки

Как отмечалось в § 8.1, диалог - это процесс обмена информацией между пользователем и программной системой, осуществляемый через интерактивный терминал и по определенным правилам.

Различают тип диалога и его форму.

Типы диалога. Тип диалога определяет, кто из «собеседников» управляет процессом обмена информацией. Соответственно различают два типа диалога: управляемые программой и управляемые пользователем.

Диалог, управляемый программой, предусматривает наличие жесткого, линейного или древовидного, т. е. включающего возможные альтернативные варианты, сценария диалога, заложенного в программное обеспечение. Такой диалог обычно сопровождают большим количеством подсказок, которые уточняют, какую информацию необходимо вводить на каждом шаге.

Диалог, управляемый пользователем, подразумевает, что сценарий диалога зависит от пользователя, который применяет систему для выполнения необходимых ему операций. При этом система обеспечивает возможность реализации различных пользовательских сценариев.

Формы диалога. Никакой диалог невозможен, если не существует языка, понятного «собеседникам». Описание языка, на котором ведется диалог, включает определение его синтаксиса - правил, определяющих допустимые конструкции (слова, предложения) языка или его форму, и семантики - правил, определяющих смысл синтаксически корректных конструкций языка или его содержание. В зависимости от вида используемых в конкретном случае синтаксиса и семантики различают три формы диалога:

- фразовую,
- директивную,
- табличную.

Фразовая форма предполагает «общение» с пользователем на естественном языке или его подмножестве. Содержание диалога в данной форме составляют повелительные, повествовательные и вопросительные предложения и ответы на вопросы. Общение может осуществляться в свободном формате, но возможна и фиксация отдельных фраз.

Организация диалога на естественном языке на современном уровне -задача не решенная, так как естественный язык крайне сложен и пока не удается в достаточной степени формализовать его синтаксис и семантику.

Чаще всего используют диалоги, предполагающие односложные ответы, например:

Программа: Введите свой возраст (полных лет):

Пользователь: 48.

В этом случае программа содержит ограниченное описание как синтаксиса, так и семантики используемого ограниченно-естественного языка. Для данного примера достаточно определить синтаксис понятия «целое положительное число» и наложить ограничение на значение числа.

Однако существует некоторый опыт создания интерфейсов на базе ограниченного подмножества предложений естественного языка в основном для интеллектуальных систем. Синтаксис и семантика языков диалога, реализуемых в таких интерфейсах, достаточно сложны.

При обработке фраз в этих случаях оперируют понятием словоформа. Словоформа - отрезок текста между двумя соседними пробелами или знаками препинания. Обработка словоформ вне связи с контекстом называется морфологическим анализом.

Выделяют два метода морфологического анализа:

• декларативный - предполагает, что в словаре находятся все возможные словоформы каждого слова, тогда анализ сводится к поиску словоформы в словаре. Данный метод обеспечивает возможность обработки сообщений, состоящих из строчных и прописных букв в произвольной комбинации, при чем как латинского, так и русского или других алфавитов;

• процедурный - предполагает выделение в текущей словоформе основы, которую затем идентифицируют.

После распознавания словоформ осуществляют синтаксический анализ сообщения, по результатам которого определяют его синтаксическую структуру, т. е. выполняют разбор предложения.

Далее выполняют семантический анализ, т. е. определяют смысловые отношения между словоформами. При этом выявляют главные предикаты, определяющие смысл предложения.

Таким образом, интерфейс, реализующий фразовую форму диалога, должен: преобразовывать сообщения из естественно-языковой формы в форму внутреннего представления и обратно, выполнять анализ и синтез сообщений пользователя и системы, отслеживать и запоминать пройденную часть диалога.

Основными недостатками фразовой формы при использовании подмножества естественного языка являются:

• большие затраты ресурсов;

• отсутствие гарантии однозначной интерпретации формулировок;

• необходимость ввода длинных грамматически правильных фраз. Основное достоинство фразовой формы состоит в относительно сво бодном общении с системой.

Директивная форма предполагает использование команд (директив) специально разработанного формального языка. Командой в этом случае называют предложение этого языка, описывающее комбинированные данные, которые включают идентификатор инициируемого процесса и, при необходимости, данные для него.

Команду можно вводить:

• в виде строки текста, специально разработанного формата, например, команды MS DOS, которые вводятся в командной строке;

• нажатием некоторой комбинации клавиш клавиатуры, например, ком бинации «быстрого доступа» современных Windows-приложений;

• посредством манипулирования мышью, например, «перетаскиванием» пиктограмм;

• комбинацией второго и третьего способов. Основными достоинствами директивной формы являются:

• сравнительно небольшой объем вводимой информации;

• гибкость - возможности выбора операции в данном случае ограничены только набором допустимых команд;

• ориентация на диалог, управляемый пользователем;

• использование минимальной области экрана или неиспользование ее вообще;

• возможность совмещения с другими формами.

Недостатки директивной формы:

• практическое отсутствие подсказок на экране, что требует запомина ния вводимых команд и их синтаксиса;

• почти полное отсутствие обратной связи о состоянии инициированных процессов;

• необходимость навыков ввода текстовой информации или манипуля ций мышью;

• отсутствие возможности настройки пользователем.

Исследования показали, что директивная форма удобна для пользователя-профессионала, который обычно быстро запоминает синтаксис часто используемых команды или комбинации клавиш. Основные достоинства формы (гибкость и хорошие временные характеристики) проявляются в этом случае особенно ярко.

Табличная форма предполагает, что пользователь выбирает ответ из предложенных программой. Язык диалога для табличной формы имеет простейший синтаксис и однозначную семантику, что достаточно легко реализовать. Удобна эта форма и для пользователя, так как выбрать всегда проще, чем вспомнить, что особенно существенно для пользователя-непрофессионала или пользователя, редко использующего конкретное программное обеспечение. Однако применение табличной формы возможно не всегда: ее можно использовать только, если множество возможных ответов на конкретный вопрос конечно. Причем, если количество возможных ответов велико (более 20), то применение табличной формы может оказаться нецелесообразным. Достоинствами табличной формы являются:

• наличие подсказки, что уменьшает нагрузку на память пользователя, так как данная форма ориентирована не на запоминание, а на узнавание;

• сокращение количества ошибок ввода: пользователь не вводит инфор мацию, а указывает на нее;

• сокращение времени обучения пользователя;

• возможность совмещения с другими формами;

• в некоторых случаях возможность настройки пользователем. К недостаткам данной формы относят:

• необходимость наличия навыков навигации по экрану;

• использование сравнительно большой площади экрана для изображе ния визуальных компонентов;

• интенсивное использование ресурсов компьютера, связанное с необхо димостью постоянного обновления информации на экране.

Следует иметь в виду, что типы и формы диалога выбирают независимо друг от друга: любая форма применима для обоих типов диалогов (рис. 8.10). Однако фразовая форма, которая используется в диалоге, управляемом пользователем, как правило, предполагает более сложные синтаксис и семантику языка диалога, так как программа должна «понимать» пользователя.

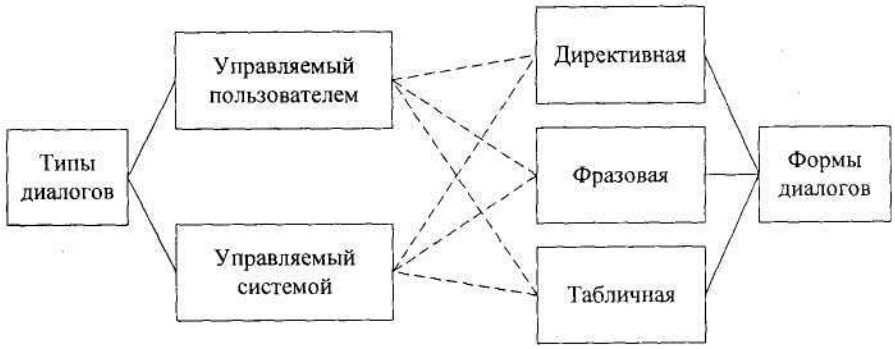


Рис. 8.10. Соответствие типов диалогов и его форм

Сложное программное обеспечение обычно взаимодействует с пользо-вателем посредством диалогов различных типов и форм в зависимости от ре-шаемых задач. Причем, помимо диалогов, происходящих в процессе нормальной работы программного обеспечения и называемых синхронными, предусматривают диалоги, возникающие по инициативе системы или пользователя при нарушении сценария нормального процесса. Такие диалоги называют асинхронными. Обычно их используют для выдачи экстренных сообщений от системы или пользователя.

Разработка диалогов. Процесс проектирования и реализации диалогов можно разделить на следующие стадии:

• определение множества необходимых диалогов, их основных сообще ний и возможных сценариев - проектирование абстрактных диалогов;

• определение типа и формы каждого диалога, а также синтаксиса и се мантики используемых языков - проектирование конкретных диалогов;

• выбор основных и дополнительных устройств и проектирование про цессов ввода-вывода для каждого диалога, а также уточнение передаваемых сообщений - проектирование технических диалогов.

В основу абстрактных диалогов должна закладываться идеология технологического процесса, для автоматизации которого предназначается программный продукт. Именно анализируя составляющие автоматизируемого технологического процесса, разработчик определяет сценарии диалогов (см. § 6.2), которые должны быть предусмотрены в программном обеспечении.

Кроме сценариев, при проектировании абстрактных диалогов используют диаграммы состоянии интерфейса или графы диалога.

Граф диалога - ориентированный взвешенный граф, каждой вершине которого сопоставлена конкретная картинка на экране {кадр) или определенное состояние диалога, характеризующееся набором доступных пользователю действий. Дуги, исходящие из вершин, показывают возможные изменения состояний при выполнении пользователем указанных действий. В качестве весов дуг указывают условия переходов из состояния в состояние и операции, выполняемые во время перехода.

Таким образом, каждый маршрут на графе соответствует возможному варианту диалога. Причем представление диалога в виде графа в зависимости от стадии разработки может выполняться с разной степенью детализации. По сути граф диалога - это граф состояний конечного автомата, моделирующего поведение программного обеспечения при воздействиях пользователя. Для представления таких графов уже были введены две нотации: нотация диаграмм состояний структурного подхода к разработке (см. рис. 4.3) и нотация диаграмм состояний UML (см. рис. 7.17). Причем нотация UML является более мощной, так как позволяет использовать обобщенные состояния. Поэтому, чтобы не вводить новую нотацию для представления графа диалога, будем использовать обозначения UML.

Пример 8.2. Разработать граф диалога для системы решения комбинаторно-оптимизационных задач.

Так как диалог на верхнем уровне должен обеспечивать реализацию диаграммы вариантов использования, исходный вариант графа диалога строим на основе анализа этой диаграммы (см. рис. 6.4). Можно предположить, что пользователь будет принимать решение о сохранении или удалении результатов после их просмотра, поэтому эти операции естественно объединить в единую группу. Кроме того, в ту же группу целесообразно добавить операцию печати результатов. Аналогично просмотр данных целесообразно объединить с их удалением или корректировкой. Операцию Новое задание целесообразно поместить в отдельную группу (рис. 8.11).

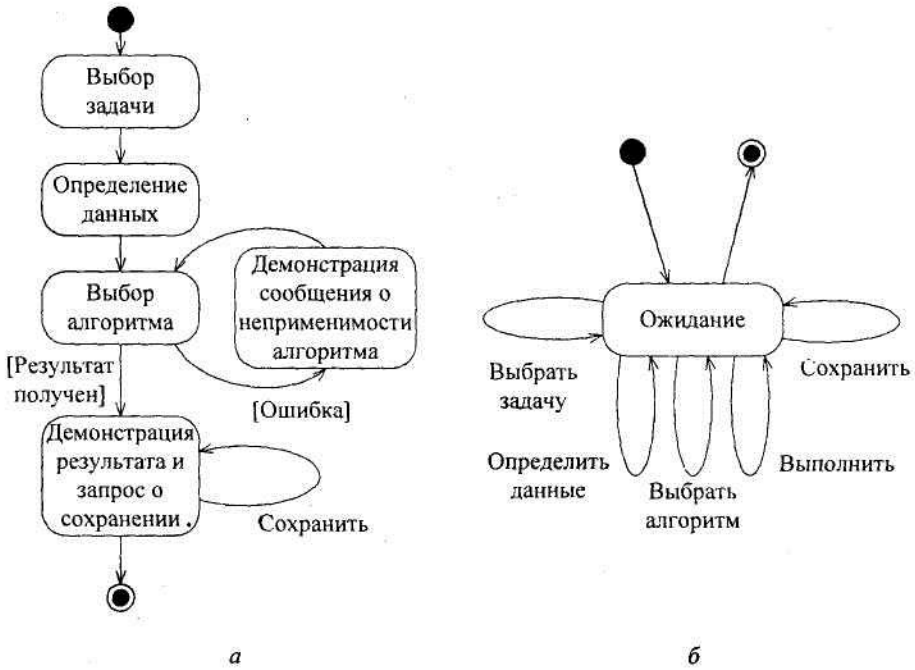
Рис. 8.11. Граф абстрактного диалога системы решения комбинаторно-оптимизационных задач

На верхнем уровне диалог очевидно должен управляться пользователем. Директивная и табличная формы могут использоваться альтернативно, по желанию пользователя, а применение фразовой формы нецелесообразно.

Пример 8.3. Детализировать диалог Новое задание.

В § 6.2 приведен сценарий Выполнения задания, на базе которого можно предложить граф диалога, управляемого системой (рис. 8.12, а). Однако этот же диалог можно представить и в виде диалога, управляемого пользователем (рис. 8.12, о).

Анализ графов диалога показывает, что диалог, управляемый системой,



а *б*

Рис. 8.12. Графы абстрактного диалога Новое задание:
а – диалог, управляемый системой; *б* – диалог, управляемый пользователем

в данном случае сильно ограничивает пользователя в выборе вариантов действия, а диалог, управляемый пользователем, предполагает выбор действия после каждого шага, хотя по смыслу эти шаги чаще всего будут выполняться последовательно.

Поэтому для реализации лучше использовать комбинированный вариант, который учитывает наличие сценария, но допускает отклонения от него по желанию пользователя (рис. 8.13).

Теперь необходимо определить, какие формы диалога можно использовать для каждого шага диалога. Первый шаг - Выбор задачи включает три варианта, поэтому имеет смысл использовать табличную форму. Второй шаг -Определение данных не конкретизирован, следовательно, уточнить его форму пока невозможно. Третий шаг - Выбор алгоритма опять же предполагает выбор, причем количество вариантов невелико: целесообразно использовать табличную форму. В остальных случаях также предпочтительной оказывается именно эта форма.

Последний этап проектирования интерфейсов - разработка конкретных операций ввода-вывода для каждого диалога с учетом специфики формы ин-

Тема 4. Факторы оценки пользовательских интерфейсов

Существует четыре основных (все остальные – производные) критерия качества любого интерфейса, а именно: скорость работы пользователей, количество человеческих ошибок, скорость обучения и субъективное удовлетворение пользователей (подразумевается, что соответствие интерфейса задачам пользователя является неотъемлемым свойством интерфейса). Эти критерии и рассматриваются в этой части книги.

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 с восторгом.

Длительность выполнения работы пользователем состоит из длительности восприятия исходной информации, длительности интеллектуальной работы (в смысле – пользователь думает, что он должен сделать), длительности физических действий пользователя и длительности реакции системы. Как правило, длительность реакции системы является наименее значимым фактором.

Критерий скорости работы удостоился определенного почета: для его оценки был выведен чуть ли не единственный в интерфейсной науке неэвристический метод, называемый GOMS «Предсказание скорости».

В 1983 году Кард, Моран и Ньювел создали метод оценки скорости работы с системой, названный аббревиатурой GOMS (Goals, Operators, Methods, and Selection Rules – цели, операторы, методы и правила их выбора).

Идея метода очень проста: все действия пользователя можно разложить на составляющие (например, взять мышь или передвинуть курсор).

Ограничив номенклатуру этих составляющих, можно замерить время их выполнения на массе пользователей, после чего получить статистически верные значения длительности этих составляющих. После чего предсказание скорости

выполнения какой-либо задачи, или, вернее, выбор наиболее эффективного решения, становится довольно простым делом – нужно только разложить эту задачу на составляющие, после чего, зная продолжительность каждой составляющей, всё сложить и узнать длительность всего процесса. Обычно тот интерфейс лучше, при котором время выполнения задачи меньше.

Впоследствии было разработано несколько более сложных (и точных) вариантов этого метода, но самым распространенным всё равно является изначальный, называемый Keystrokelevel Model (KLM). К сожалению, этот вариант метода имеет определенные недостатки:

- он применим в основном для предсказания действий опытных пользователей;

- он никак не учитывает ни прогресса в обучении, ни возможных ошибок, ни степени удовлетворения пользователей;

- он плохо применим при проектировании сайтов из-за непредсказуемого времени реакции системы.

Для его использования достаточно знать правила разбиения задачи на составляющие и длительность каждой составляющей (рекомендую на первое время повесить у себя на рабочем месте листок с цифрами).

Правила GOMS

- ✓ Нажатие на клавишу клавиатуры, включая Alt, Ctrl и Shift (К): 0,28 сек
- ✓ Нажатие на кнопку мыши (М): 0,1 сек
- ✓ Перемещение курсора мыши (П): 1,1 сек
- ✓ Взятие или бросание мыши (В): 0,4 сек
- ✓ Продолжительность выбора действия (Д): 1,2 сек.
- ✓ Время реакции системы (Р): от 0,1 сек до бесконечности. Для базовых операций, таких как работа с меню, это время можно не засчитывать, поскольку с момента создания метода производительность компьютеров многократно возросла.
- ✓ Методика расчетов Предположим, от пользователя со средним опытом требуется сохранить в активном каталоге текущий документ под именем Опись и выйти из программы. Подразумевается, что диалоговое окно сохранения файла выглядит следующим образом:

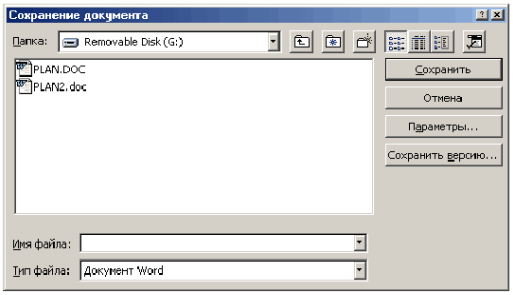


Рис. 67. Диалоговое окно сохранения файла в Word. © Microsoft.

Эта задача состоит из следующих составляющих:

Тип действия	Продолжительность	Комментарий
Д	1,2	Пользователь анализирует свою задачу и создает алгоритм её решения
П	1,1	Перемещение курсора к меню Файл
М	0,1	Нажатие кнопки мыши
Д	1,2	Открылось меню и пользователю необходимо найти нужный элемент (и понять, какую именно команду он должен выбрать: Сохранить или Сохранить как...)
П	1,1	Перемещение курсора к элементу меню Файл: Сохранить как...
М	0,1	Нажатие кнопки мыши
Р	0,1	Время реакции системы – открывается диалоговое окно сохранения файла
Д	1,2	Открылось диалоговое окно сохранения файла. От пользователя требуется рассмотреть его и понять, что именно ему нужно сделать
П	1,1	Перемещение курсора к полю ввода названия файла
М	0,1	Нажатие кнопки мыши для перемещения фокуса ввода
Д	1,2	Пользователь выдумывает файлу название (но мы-то знаем, что он лишен свободы воли и назовет файл словом Опись)
В	0.4	Перенос руки с мыши на клавиатуру

Итого на эту операцию пользователю потребуется 16,08 секунд. Предположим теперь, что ту же самую операцию выполняет продвинутый пользователь, знающий, что если закрыть программу с помощью пиктограммы в её титульной строке, имея несохраненный документ, то программа сама предложит его записать:

Тип действия	Продолжительность	Комментарий
Д	1,2	Пользователь анализирует свою задачу и создает алгоритм её решения
П	1,1	Перемещение курсора к меню элементу окна Закрыть
М	0,1	Нажатие кнопки мыши
Р	0,1	Время реакции системы – открывается диалоговое окно сохранения файла
Д	1,2	Открылось диалоговое окно сохранения файла. От пользователя требуется рассмотреть его и понять, что именно ему нужно сделать
П	1,1	Перемещение курсора к полю ввода названия файла
М	0,1	Нажатие кнопки мыши для перемещения фокуса ввода
Д	1,2	Пользователь выдумывает файлу название
В	0.4	Перенос руки с мыши на клавиатуру
К х 7	0,28 х 7	Ввод названия файла. В придачу к шести изначальным нажатиям, пользователь нажимает клавишу Enter, сразу инициируя запись файла

Итого 8,56 сек. Чуть ли не вдвое меньше. Второй вариант при прочих равных условиях эффективнее. Все и раньше это знали, но зато теперь у нас есть научное доказательство.

Длительность интеллектуальной работы

Согласно Дональду Норманну, взаимодействие пользователя с системой (не только компьютерной) состоит из шести 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. Разберем это подробнее.

Непосредственное манипулирование

Как уже было сказано, перед действием пользователи «проявляют тенденцию думать». В процессе этого думанья им приходится из общего, еще неконкретного замысла формировать четкую последовательность действий. Что нелегко.

Предположим, пользователь чайника хочет выпить чаю. Желание выпить чаю есть цель действий. Осознав её, пользователь формирует общий замысел, а именно «А вот неплохо бы поставить чайник и устроить себе чаю». После этого пользователь строит алгоритм своих действий:

Подойти к чайнику и открыть крышку. Если воды в чайнике мало или нет вовсе, перенести чайник к раковине и наполнить его водой, после чего поставить его на плиту. Если воды в чайнике достаточно, сразу поставить его на плиту. Закрыть чайник крышкой. Найти спички. Открыть коробок, вытащить одну спичку, закрыть коробок, зажечь спичку. Спичкой зажечь под чайником газ, установив подачу газа на максимум. Потушить спичку и выкинуть её. Подождать, пока чайник не закипит, в это время найти достаточно чистый стакан и налить в него заварки. По желанию, найти сахарницу и добавить сахару в стакан. Выключить газ. Налить кипятка из чайника в стакан.

Размешать жидкость мизинцем (время от времени вытаскивая его из жидкости и дуя на него, чтобы не обжечься). Употребить жидкость по назначению. Ах, да. Закрыть кран в раковине.

Разумеется, в реальной жизни такую сложную программу пользователь не создает – как-никак, он обустраивал себе чай несколько тысяч раз, действие успело стать автоматическим и создаваемый алгоритм состоит в лучшем случае из элементов высшего порядка (поставить чайник, налить чаю).

В случае же компьютерных систем трудно ожидать такого автоматизма, более того, алгоритмы действий всегда получаются слишком абстрактными (а люди плохо справляются с абстракциями).

Анализируя пример с чаем, можно выделить определенные требования к человеку, выполняющему работу.

Он должен знать:

1 что он хочет получить на выходе (чай)

2 как минимум одну последовательность действий, приводящую к успешному результату (наполнить чайник, поставить его на плиту, дождаться закипания, налить кипяток в стакан с заваркой)

3 где ему найти все объекты, участвующие в процедуре (где, черт побери, спички?)

4 как определять годность объектов к использованию (есть ли вода в чайнике)

5 как управляться с объектами (как включить газ).

Список, как видим, довольно внушительный. И если с первым пунктом проблем обычно не возникает, то с остальными приходится повозиться. Плохая новость заключается в том, что остальных пунктов много, хорошая новость – в том, что решение всех этих проблем единое. Оно называется непосредственным манипулированием (direct manipulation).

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!»1. 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 на 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. Чтобы лучше оценить прелесть этого метода, удобно сравнить три варианта действий пользователя на примере этого самого стирания:

Выбор команд из меню	Использование горячих клавиш	Использование элемента на панели инструментов	Непосредственное манипулирование
Формирование цели действий и общего замысла			
Определение необходимых действий и их последовательности			

Выбор файла			
Поиск меню, ответственного за стирание	Поиск в памяти команды стирания	Поиск на экране соответствующей пиктограммы	Поиск корзины
Поиск элемента меню, вызывающее стирание файла	Поиск клавиши Delete на клавиатуре	Нажатие на пиктограмму	Перенос файла в корзину
Выбор нужного элемента меню	Нажатие клавиши Delete		

Видно, что даже такое простое действие, как стирание файла, на самом деле состоит из многих малых, уже не делимых, действий (атомов). При этом для ускорения мыслительной работы пользователя необходимо не только сокращать количество этих атомов, но и делать эти атомы более простыми. Первые три атома у любого метода одинаковы, тут уж ничего не придумать. Различие между методами только в конце процедуры.

Из таблицы сразу видно, что метод выбора команды из меню плох уже тем, что состоит из большого количества атомов. С другой стороны, он имеет то достоинство, что пользователь, вообще ничего не знающий о системе, только лишь благодаря сканированию меню может узнать, что файлы вообще *можно* стирать (собственно говоря, эта *обучающая функция* составляет главное достоинство меню как метода взаимодействия пользователя с системой, об этом подробнее см. «Меню» на стр. 77). Но поскольку это достоинство не имеет прямого отношения к скорости работы, можно смело сказать, что метод выбора команд из меню из состязания выбыл.

Количество элементов второго метода, использующего горячую клавишу, также велико, но у него есть определенные плюсы. При достаточной степени автоматизма нет ни необходимости искать клавишу на клавиатуре, ни думать, какую клавишу нажать. Таким образом, для опытных пользователей этот метод очень хорош.

Третий способ, нажатие на кнопку в панели инструментов, состоит из не столь большого количества элементов, так что формально он хорош. К сожалению, он не слишком универсален. Количество элементов в любой панели инструментов ограничено, так что особенно с этим способом не развернешься. Не говоря уже о том, что для многих действий невозможно подобрать пиктограмму. В то же время способ этот имеет одно существенное достоинство – подсказка к действию постоянно находится на экране, так что пользователю не приходится копаться в своей памяти (что может быть очень долгим).

И, наконец, четвертый способ– непосредственное манипулирование. Помимо того, что он сам по себе состоит из небольшого количества атомов, 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.

И еще раз более того. Предположим, что пользователь собрался стереть важный системный файл, который стирать нельзя. Методы выбора команды в меню и в панели инструментов, равно как и метод непосредственного манипулирования, здесь сработают – элемент можно будет превентивно заблокировать. Если же пользователь попытается стереть файл, нажав на Delete, система окажется неспособна как-то показать неправомочность его действий (разве что писком или сообщением об ошибке, что нехорошо, см. «Вон отсюда, идиот!» на стр. 41). А теперь предположим, что пользователь собрался стереть важный файл, который стирать не рекомендуется. Ни один методов, кроме непосредственного манипулирования (можно будет поменять пиктограмму корзины на время, пока курсор, с зажатым в него файлом, будет находиться над ней), здесь не сработает, т.е. этот метод отличается от остальных своей гибкостью.

Важно понимать еще две вещи. Во-первых, для достижения достаточной эффективности не обязательно стараться наиболее реалистично отразить действие, значительно важнее возможно более реалистично отразить объект, над которым это действие совершается. Например, компьютерную панель управления работой осветительных приборов необязательно снабжать точными имитациями выключателей. Главное реалистично отразить на ней план помещения и расположение источников света, равно как и показать прямую (читай – непосредственную) связь между этой информацией и собственно выключателями. Во-вторых, бывают ситуации, когда эффективность непосредственного манипулирования уравновешивается неэффективностью физических действий пользователя.

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.

Телефонный звонок, обеденный перерыв, анекдот, рассказанный коллегой. В течение работы происходит множество таких отвлечений. Помимо них пользователя отвлекает множество мелочей: листок бумаги на столе перекрывает другой, нужный; мимо кто-то проходит и нужно бросить на него беглый взгляд, не коварный враг ли приближается, чтобы задушить.

Более того, даже посторонняя мысль также отвлекает от работы, например я, пока писал этот абзац, отвлекался 14 раз (я считал разы, это число не случайно).

Но это еще не главное: каждый раз, когда пользователь прерывает свою деятельность и начинает думать о том, что ему делать дальше, он отвлекается тоже. И эти раздумья отвлекают пользователя значительно чаще, чем всё остальное.

Каждое такое отвлечение занимает определенное время. Хуже того, оно сбивает фокус внимания, т.е. обработку текущего действия. После каждого такого отвлечения пользователь должен либо вспоминать текущую задачу, либо заново её ставить перед собой (занимает это несколько секунд, что много). Дело в том, что у человека есть только один фокус внимания, так что при любом отвлечении (которое есть не что иное, как переключение на другую задачу) старый фокус внимания теряется. Было бы еще ничего, если бы возвращение фокуса требовало только изменения направления взгляда.

Но при отвлечении новые стимулы заменяют содержимое кратковременной памяти (см. стр. 48), так что для возвращения к работе от пользователя требуется заново поместить в свою память нужную информацию.

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, об этом, собственно, вся эта книга. Так что эта глава только 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.

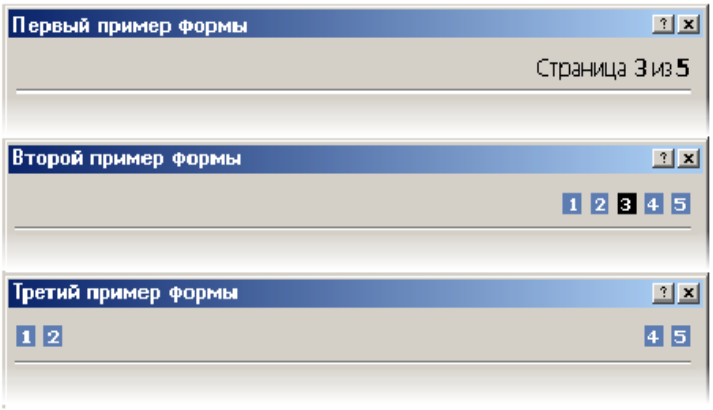


Рис. 1. Три варианта индикации степени заполнения экранной формы. Второй и третий варианты значительно визуальнее. Не используйте в подобных случаях ползунки (см. стр. 76).

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

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, чем точнее должно быть движение, тем более плавным и замедленным оно должно быть. 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)1 сформулировал правило, впоследствии ставшее известным как Закон Фитса: Время достижения цели обратно пропорционально размеру цели и дистанции до цели. 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 (ну и точности выбора начального направления движения). Понятно, что кнопка, расположенная в углу экрана, имеет «еще более бесконечные» размеры, если так вообще можно сказать (т.е. 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 минимально1. 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 контекстно-зависимы, 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 лучше2, 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 (по GOMS, см. «Предсказание скорости» на стр. 122), 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 говорит обратное). 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, об этом см. «Перемещение в пределах окна» на стр. 98

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 в туалет, 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.

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.

И обратно – индикатор показывает, что процесс выполняется очень быстро. Пользователь понимает, что у него есть всего минута и в спешке убегает в другую конату. Возвратившись, он обнаруживает, что индикатор застрял на двадцати процентах и не проявляет тенденции снова быстро расти.

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, выполняющихся с разной скоростью, можно не утруждаться, стараясь так сбалансировать рост индикатора, чтобы он всё время происходил с одинаковой скоростью.

Иногда это «неутруждение» принимает довольно комичные формы, так, однажды я видел индикатор выполнения, который сначала рос, потом стал снижаться, потом опять вырос. Проблема в том, что пользователи рассматривают такие индикаторы именно как способ узнать, когда процесс завершится. Так что врать пользователю тут нехорошо.

Тема 5. 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 операционистка в банке может сделать кого-то богаче, а банк, в свою очередь, беднее (впрочем, обычно беднее становятся все).

Существование несуществующего

Вначале необходимо сказать главное: они не существуют. Компьютеры (как и все сложные технические системы) вообще не могут быть используемы человеком без совершения 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.

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.

Под словосочетанием «человеческая ошибка» будем иметь в виду только действие пользователя, не совпадающее с целью действий этого пользователя.

Типы ошибок

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 пользовательской документации не бывает.

Опечатки. «Опечатки» происходят в двух случаях: во-первых, когда не все внимание уделяется выполнению текущего действия (этот тип ошибок характерен, прежде всего, для опытных пользователей, не проверяющих каждый свой шаг) и, во-вторых, 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).

Несчитывание показаний системы. Ошибки, 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 это делать).

Моторные ошибки. 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.

В действительности надо стремиться минимизировать количество ошибок, поскольку только это позволяет сберечь время (т.е. повысить производительность). Суммируя, при борьбе с ошибками нужно направлять усилия на:

1) плавное обучение пользователей *в процессе* работы;
2) снижение требований к бдительности;
3) повышение разборчивости и заметности индикаторов;

Дополнительно к этим трём направлениям, есть и четвертое: снижение чувствительности системы к ошибкам.

Для этого есть три основных способа, а именно:

1) блокировка потенциально опасных действий пользователя до получения подтверждения правильности действия;

2) проверка системой всех действий пользователя перед их принятием;

3) самостоятельный выбор системой необходимых команд или параметров, при котором от пользователя требуется только проверка.

При этом 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. Проиллюстрирую эту проблему на примере.

Некто Виктор Х. хочет удалить файл День рождения. Он выделяет этот файл и отдает системе команду Удалить. Появляется диалоговое окно с требованием подтвердить удаление файла. Начинается самое интересное. Виктор Х., не глядя на диалог, знает, зачем он нужен. Он видел его не раз и не два. Он даже дословно помнит, что именно его спрашивают. «Да, хочу» говорит Виктор Х. и нажимает кнопку ОК. После чего рвет и мечет, поскольку вместо файла День рождения он стер файл Пароль от сейфа.

Проблема появилась в самом начале, потому что он выбрал не тот файл. Так с Виктором Х. было не всегда. Когда он только учился пользоваться компьютером, каждое открывшееся диалоговое окно наполняло его сердце ужасом. От этого ужаса он читал тексты на всех диалоговых окнах и благодаря этому мог вовремя остановиться и не стереть нужный ему файл. Что всё это значит – 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 получит 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, есть еще и невидимые данные.

Не делайте опасные для пользователя кнопки кнопками по умолчанию

Также к этому типу блокировки относится снятие фокуса ввода с терминационных кнопок, чтобы пользователь не мог, не разобравшись, нажать на 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. Важно только не делать кнопку кнопкой по умолчанию и кнопку Отмена (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. Я знаю только 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.

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.

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 (spinner). Фактически это обычное поле ввода, снабженное двумя кнопками для модификации его содержимого (в сторону уменьшения и

увеличения). Интересен 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

Если пользователь ввёл некорректное число с клавиатуры, нужно 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).

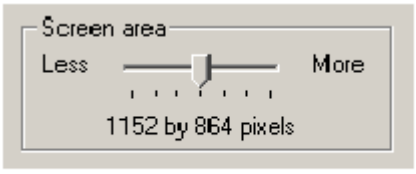


Рис. 3. Ползунок.

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. Вопрос состоит в том, как системе узнать, что именно нужно пользователю.

Проиллюстрировать сферу применения данного метода удобно на примере печати.

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.

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 по тем или иным причинам невозможно обнаружить формальной проверкой (т.е. невозможно обнаружить их не случайно). Пример: грамматическая ошибка в тексте, формально удовлетворяющем правилам языка.

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, при этом сказать, какое сильнее, затруднительно. При этом объяснения еще и складываются. Но, по порядку.

Объективное объяснение просто: ошибки, исправляемые после, снижают производительность работы. Как мы уже знаем, 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 системой.

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.

Тема 6. 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 производится.

Всё это делает проблему обучения пользователей работе с компьютерной системой чрезвычайно важной. Начиная с определенного объема функциональности системы, количество пользователей, знающих *всю функциональность*, неуклонно снижается.

Т.е. чем объемней система, тем больше шансов на то, что среднестатистический пользователь знает о ней очень немного (относительно общего объема функциональности).

Так, я уверен, что на свете нет ни единого человека, который бы *полностью* знал MS Word, предполагаю, что средний пользователь умеет пользоваться не более чем пятью процентами его возможностей.

Плохо это по многим причинам:

во-первых, пользователи работают с системой не слишком эффективно, поскольку вместо методов адекватных они используют методы знакомые.

во-вторых, достаточно часто случается, что пользователи, не зная, что имеющийся продукт делает то, что им нужно, ищут (и находят) продукт конкурента.

в-третьих, при таком положении вещей затруднительно продавать новые версии продукта: если пользователь не умеет пользоваться и тем, что есть, убеждать его совершить покупку ради новой функциональности придется на довольно шатком фундаменте.

Почему пользователи учатся

Есть непреложный закон природы: люди делают что-либо только при наличии стимула, при этом тяжесть действия пропорциональна силе стимула.

Обучение есть действие: если обучаться легко, пользователям будет достаточно слабого стимула, если тяжело, стимул придется увеличивать.

Пользователь обучится пользоваться программой или сайтом только в том случае, если он будет уверен, что это сделает его жизнь легче и приятней. На профессиональном жаргоне это называется возвращением инвестиций (return of investments, ROI): ни один инвестор не вложит деньги (действие) без уверенности, что эти деньги принесут ему доход (стимул), если же полной уверенности в этом нет, ожидаемая прибыль должна быть огромна. Это правило в полной мере касается и пользователей.

Есть ещё одно правило: 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

Таким образом, чтобы пользователь начал учиться, ему нужно рассказать о функциональности системы. А дальше пользователь сам будет учиться, если, конечно, стимул достаточен.

Без этого любая система не вызовет никакого желания учиться, даже если это обучение принесло бы пользователю множество пользы. Простота же обучения системе есть всего лишь метод уменьшить «необходимый и достаточный» стимул; при достаточно сильном стимуле люди охотно учатся и без всякой простоты (другой разговор, что получение достаточно весомого стимула часто более сложно для дизайнера, чем облегчение обучения).

Средства обучения

более совершенный список средств обучения:

- ✓ общая «понятность» системы
- ✓ обучающие материалы.

А теперь можно разобрать эти составляющие по отдельности.

Понятность системы

Термин «понятность» включает: ментальную модель, метафору, аффорданс и стандарт.

Ментальная модель

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.**

Разберем её на примере утюга. Утюгом никогда не сможет воспользоваться человек, который не знает, что провод от утюга надо воткнуть в розетку. Но, обладая таким знанием, человек может пользоваться утюгом, не зная, сколько энергии утюг потребляет (отсутствие точности), равно как сохраняя искреннюю уверенность, что по проводам, как вода, течёт электричество (отсутствие правильности). Беда приходит тогда, когда представления человека о системе концептуально не совпадают с реальным устройством системы.

Таким образом, без корректной ментальной модели пользователи фактически неспособны научиться пользоваться системой. К сожалению, проектирование системы, для которой модель построить легко, есть дело сложное, так что придумать для него универсальный алгоритм невозможно.

Единственно, что может помочь, это творческий подход к проектированию.

Существует одно простое правило: поскольку элементы, выполняющие несколько разных функций в зависимости от контекста, существенно усложняют построение ментальной модели, их лучше не создавать.

Поэтомулучше делать слишком много элементов, нежели слишком мало.

Метафора

Как было сказано, чтобы научиться пользоваться системой, пользователю нужно построить ментальную модель этой системы. Чтобы избавить его и от этой работы, нужно добиться применения метафоры, которая позволяет пользователю не создавать новую модель, а воспользоваться готовой моделью, которую он ранее построил по другому поводу.

Самым простым примером метафоры в интерфейсе является устройство программ для проигрывания звуков на компьютере. Исторически сложилось, что вся аудиотехника имеет почти одинаковый набор кнопок:

несколько кнопок со стрелками (назад/вперед), кнопка с треугольником (воспроизведение), кнопка с двумя дощечками (пауза), кнопка с квадратиком (полная остановка) и красный кружок (запись). Про них нельзя сказать, что они совершенно понятны, но научиться им можно без труда.

При этом обычно жизнь складывается так, что сначала человек научается пользоваться этими кнопками на материальных устройствах, а уж потом начинает пользоваться компьютером.

Соответственно, при проектировании программы аналогичного назначения разумно скопировать существующую систему маркировки кнопок. Благодаря этому пользователям для использования программы ничему не приходится учиться (и даже не приходится переучиваться, что вдвойне обидно, поскольку полностью отрицает возвращение инвестиций в обучение).

Недостатки метафор.

Во-первых, не для любой функциональности можно подобрать подходящую метафору, причем заранее узнать, есть ли хорошая метафора или нет, невозможно, так что можно потратить время на поиски и ничего не найти. Это, как минимум, неэффективно.

Во-вторых, даже подходящая метафора может оказаться бесполезной, если её не знает существенная часть аудитории или её тяжело однозначно передать интерфейсом.

В-третьих, почти всегда метафора будет сковывать функциональные возможности. Что делать, если проектируемая система обладает большим количеством функций, чем копируемый образец? Следование метафоре в таких условиях будет только вредить, поскольку совпадающим функциям будет учиться легче, а несовпадающим – сложнее (они будут слишком иначе устроены). Зачем тогда система, почему бы пользователю не воспользоваться её исходным образцом?

В-четвертых, совершенно необязательно, что сам по себе копируемый образец работает идеально. Если его копировать, окажется, что система не сможет быть эффективней своего прародителя. Например, Adobe PageMaker во многом копирует традиционные верстальные гранки, наследуя их известность пользователям вместе с их ограничениями. Благодаря этому он стал чрезвычайно популярен.

В-пятых, почти всегда метафору можно использовать в документации, не перенося её в интерфейс, при этом с тем же успехом. Достаточно просто написать, что «система во многом напоминает …» и нужный результат будет достигнут.

Таким образом, метафора, будучи *лучшим* средством для избавления пользователя от обучения, не является средством *хорошим*. С другой стороны, метафоры иногда всё-таки работают (взять те же музыкальные программы), так что определенную пользу от них получить можно.

Анализируя опыт успешных случаев их применения, 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.

Суммируя, можно сказать, что применять метафору можно, но с большой осторожностью.

Аффорданс.

В современном значении этого термина аффордансом называется ситуация, при которой 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.

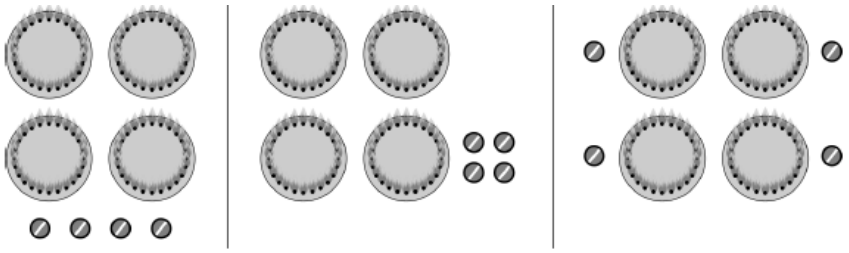


Рис. 6. Отсутствие аффорданса в дизайне кухонной плиты. Слева стандартный вариант, недостаток которого заключается в том, что невозможно умозрительно определить, какой диск управляет какой конфоркой. В центре и справа варианты с аффордансом, не имеющие этой проблемы, при этом работающие по-разному. В центральном примере расположение регуляторов повторяет расположение рабочих объектов (конфорок), благодаря чему неоднозначность исчезает. В правом примере каждому объекту соответствует отдельный регулятор. Эти два способа уничтожения неопределенности являются основными в экранном дизайне.

Проблема в том, что аффорданс на экране получить сложнее, нежели в предметах реального мира, поскольку единственным способом его передачи оказывается визуал, а такие способы, как тактильные свойства или приспособленность к человеческой анатомии (пистолет, например, *трудно* держать неправильно) оказываются за бортом. Это ограничение приводит к тому, что доступными оказываются всего несколько способов передачи аффорданса, из которых самыми значительными являются четыре:

- ✓ 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).

В целом, создание аффордансов является наиболее сложной задачей, стоящей перед графическим дизайнером, работающим над интерфейсом.

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.

Таким образом, чтобы стандарт заработал, он должен быть популярен. Популярен же он может быть 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.

Обучающие материалы

Подсистемы справки, необходимые для того, чтобы пользователь научился пользоваться системой:

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.

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.

Сообщения об ошибках.

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.

Спиральность

В отличие от художественной литературы, справочные системы не предназначены для того, чтобы приносить удовольствие, 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.

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. Исследования1 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 докомпьютерной реальности. Во многом это правильно: кто бы что ни говорил, но массовые представления о прекрасном не выросли за последние сто лет. 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. Отдельные детали стула, например, не значат особенно много, гораздо большее значение имеет впечатление от всего стула целиком. Напротив, интерфейс состоит из отдельных деталей, каждая из которых действует сравнительно независимо, поскольку раскрывает различную функциональность. Это сближает дизайн интерфейса с типографикой и 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 стульев на мебельной фабрике ограничен не только и не столько своей фантазией, но и технологией, наличием тех или иных деталей на складе, стоимостью конструируемого стула и многими другими факторами. Подобно ему, 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 дизайн, коммуникационный дизайн и промышленный дизайн. Итак, какие их принципы могут быть использованы в дизайне интерфейса?

Конструируемый предмет должен быть незаметен в процессе его использования. Странно интересоваться, как выглядит стул, на котором сидишь. Когда человек читает книгу, он чаще всего не замечает её верстку. В то же время

предмет должен приятно ощущаться на бессознательном уровне (применительно к стулу это явление можно охарактеризовать как «радость зада»). Для этого:

_ Избегайте развязности в визуале. Лучше быть поскромнее.

Во что бы то ни стало, добивайтесь неощущаемости интерфейса

_ 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 визуале.

_ Старайтесь сделать визуал максимально более легким и воздушным.

_ Старайтесь добиваться контраста не сменой насыщенности элементов, но расположением пустот.

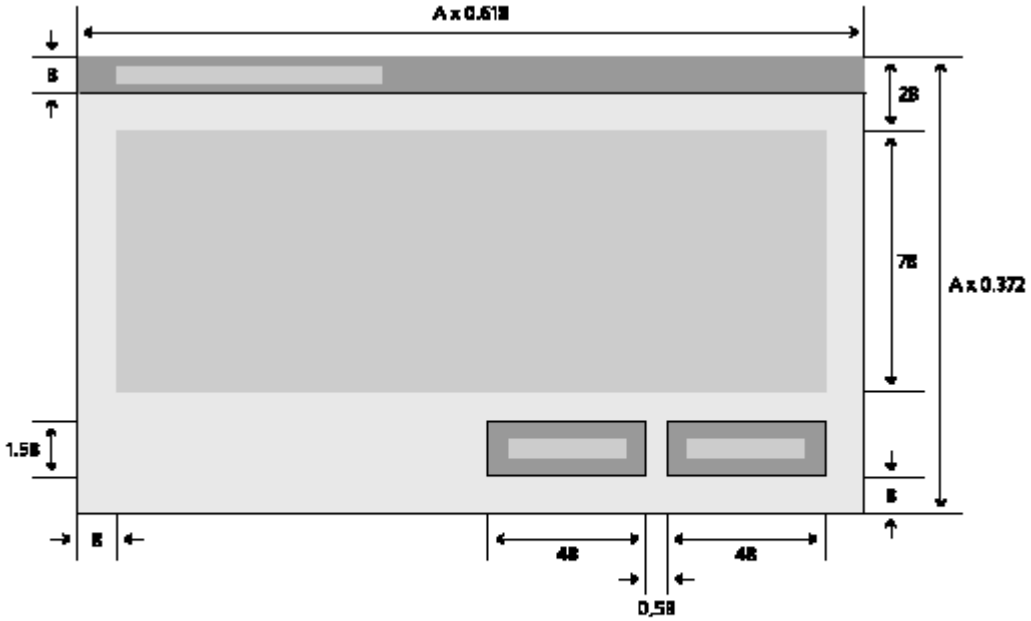


Рис. 10. Пример закономерностей в диалоговом окне. Старайтесь минимизировать количество констант (тем более, что двух констант обычно хватает на все). Разумеется, единожды примененных закономерностей необходимо придерживаться во всей системе.

Красота понятие относительное1. 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 закономерностью 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 x 0.382). Вроде бы чепуха, но результат существеннейший. Разумеется, на эту тему можно сказать очень много (странно было бы ожидать, что мне на паре страниц удастся выразить весь опыт дизайнерской культуры). Поэтому очень рекомендую прочесть несколько книг по графическому дизайну (лучше книжному, я, например, будучи еще и книжным дизайнером, не нашел *ничего* из этого моего опыта, что не было бы полезно в дизайне интерфейсов).

Какой пользователь не любит быстрой езды?

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 предыдущей части книги (см. «Скорость выполнения работы» на стр. 5) 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 – 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 продолжительность 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 действий1. 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 самим пользователем.

Функции, работающие в автоматическом режиме, но время от времени просыпающиеся и требующие от пользователей реакции, вызывают стресс. В любом случае, стоит всеми силами внушать пользователем мысль, что только явно выраженное действие приводит к ответному действию системы (это, в частности, главный аргумент против ролловеров2 – 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 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.

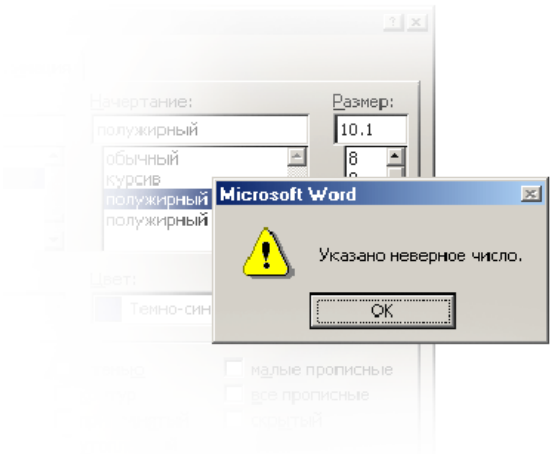


Рис. 11. Вот что бывает, если пользователь попытается ввести значение, которое ему нужно, но которое система не умеет обрабатывать. Тут возможно три альтернативных решения. Во-первых, при проектировании системы можно более тщательно подходить к выбору её функциональности. Во-вторых, можно просто игнорировать неправильность значения, округляя его до ближайшего возможного (индицируя, возможно, самостоятельность действий системы однократным миганием соответствующего поля ввода). В-третьих, вместо обычного поля ввода можно использовать крутилку (см. стр. 75). © Microsoft.

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

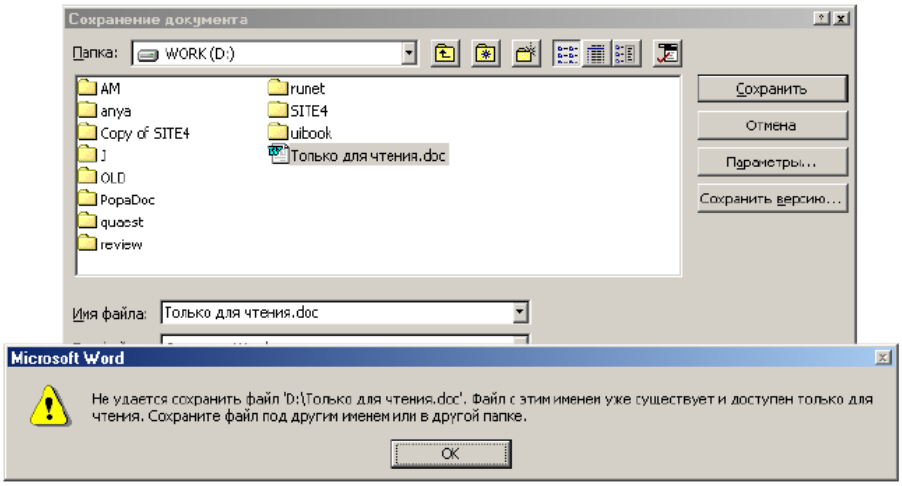


Рис. 12. Типичное сообщение об ошибке, вызванное нежеланием системы показать пользователю границы его действий. С одной стороны, оно разумно – файл не может быть записан под этим именем. С другой стороны, это сообщение показывается пользователю каждый раз при попытке перезаписать такой файл. Если бы 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.

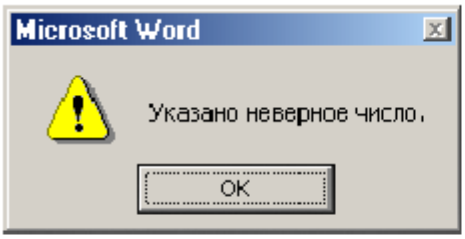


Рис. 13. Для кого неверное? И кто, собственно, виноват, система или пользователь?
© Microsoft.

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, они их ненавидят.

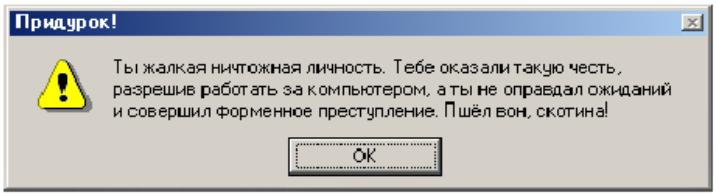


Рис. 14. Именно так пользователи воспринимают любые сообщения об ошибках.

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 сообщения 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 чрезвычайно редки, так что их можно не учитывать. Главный враг – компакт-диск.

Тут возможно несколько непротиворечащих 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:

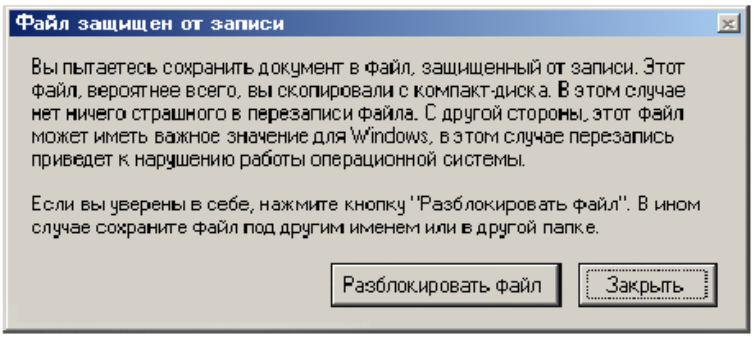


Рис. 15. Улучшенное сообщение об ошибке. Обратите внимание, что кнопка Закрыть выбрана по умолчанию, чтобы снизить вероятность перезаписи важных файлов. Конечно, лучше всего было бы, чтобы ОС сама снимала с копируемых с компакт-диска файлов метку Read Only. Многие проблемы при этом бы исчезли, поскольку 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, значительно лучше предназначенный для показа сообщений. Называется этот элемент весьма поэтично: пузырь (balloon, см. рис. 16).

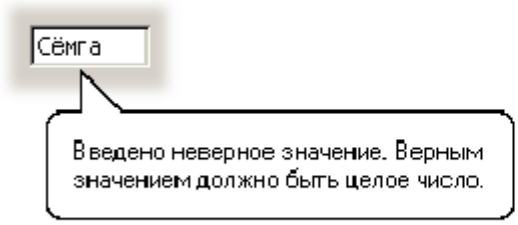


Рис. 16. Пузырь в его лучшем проявлении (не обращайте внимания на текст).

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

Тема 7. Различные элементы управления

Кнопки

Кнопкой называется элемент управления, всё взаимодействие пользователя с которым ограничивается одним действием – нажатием. Эта формулировка, кажущаяся бесполезной и примитивной, на самом деле довольно важна, поскольку переводит в гордое звание кнопок многие элементы управления, которые как кнопки по большей части не воспринимаются (об этом позже).

Командные кнопки

Нажатие на такую кнопку запускает какое-либо явное действие, поэтому правильнее называть такие кнопки «кнопками прямого действия». С другой стороны, из-за тяжеловесности этого словосочетания им всегда пренебрегают.

Рис. 30. Всё это командные кнопки, они же кнопки прямого действия (включая гипертекстовую ссылку справа).

В интернете кнопка должна быть оформлена как текстовая ссылка,
если она перемещает пользователя на другой фрагмент контента,
и как кнопка – если она запускает действие

Размеры и поля

Чем больше кнопка, тем легче попасть в нее курсором. Одновременно пользователю должно быть трудно нажать не на ту кнопку. Добиться этого можно либо изменением состояния кнопки при наведении на неё курсором, либо установлением пустого промежутка между кнопками.

Текст и пиктограммы.

Все руководства по разработке интерфейса с изумительным упорством требуют снабжать командные кнопки названиями, выраженными в виде глаголов в форме инфинитива (Прийти, Увидеть, Победить).

Помимо текста, на кнопках можно выводить пиктограммы. Эта возможность редко используется в ПО, но очень широко в интернете. Формально, на таких кнопках пиктограммы не очень хороши из-за того, что они обычно должны передавать пользователям идею *действия* (т.е. глагол), а действие плохо передается пиктограммами.

Пиктограммы хороши для тех кнопок, для которых пиктограммы нарисовать легко, и для тех кнопок, которые нужны особенно часто (при этом качество пиктограммы особого значения не имеет, важно только различия пиктограмм между собой). С другой стороны, единство и согласованность интерфейса требует, чтобы если уж есть пиктограммы, то уж везде; если же это невозможно, лучше вообще изъять пиктограммы из кнопок, поскольку их эффект невелик, а трудозатраты, уходящие на их создание, значительны.

Кнопки доступа к меню

Существует много ситуаций, когда раскрывающийся список не помещается в отведенное для него место, поскольку текст в списке слишком велик. Первое, что приходит в голову, это вставить кнопку, нажатие на которую будет вызывать меню. Рассмотрим недостатки такого решения:

Во-первых, недостаток кнопки будет проявляться в том, что, поскольку, по условиям задачи, текст не будет виден, значение кнопки будет менее понятным, чем контекстное меню безо всякой кнопки.

Во-вторых, само использование кнопки в таком исполнении не совсем правильно, поскольку нарушается принцип единообразия: пользователь нажал на кнопку, а действия как такового и нет (не считать же действием появление меню). В интернете это еще проходит, поскольку там кнопки могут и не выглядеть как кнопки, будучи оформлены как ссылки; в этом случае противоречия не возникает.

Суммируя, можно смело сказать, что использовать кнопку для инициирования показа меню можно, но стыдно. Не высший класс.

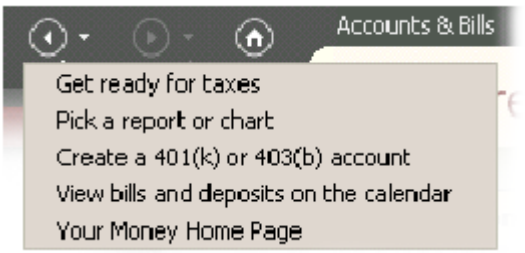


Рис. 33. Пример очень хорошего использования кнопки доступа к меню. Большинство пользователей в большинстве случаев не думают ни о каком меню, сразу нажимая на кнопку и инициируя действие. В то же время они получают возможность без труда сэкономить несколько нажатий, если обратятся к меню.

Существуют определенные ситуации, когда такие кнопки очень хороши. Для этого только нужно сделать так, чтобы кнопка была одновременно и командной кнопкой, и показывала меню. Для этого нужно сделать две вещи:

- Во-первых, нужно разделить кнопку на две области, одна из которых запускает действие, а другая открывает меню.

Во-вторых, нужно организовать такой контекст, при котором результат нажатия на кнопку всегда будет понятным.

Например, это очень хорошо работает с кнопками **Вперед и Назад**.

Осталось сказать немного. Во-первых, на области, вызывающей меню, обязательно должно находиться изображение направленной вниз стрелки.

Во-вторых, эта область должна находиться справа на кнопке, чтобы изображение стрелки не мешало воспринимать текст или пиктограмму на кнопке.

Чекбоксы и радиокнопки

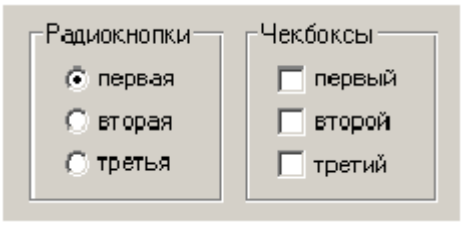


Рис. 34. Пример радиокнопок и чекбоксов.

Они являются кнопками отложенного действия, т.е. их нажатие ни при каких обстоятельствах не должно инициировать какое-либо немедленное действие. С их помощью пользователи вводят параметры, которые скажутся после, когда действие будет запущено иными элементами управления.

Нарушать это правило нельзя ни при каких обстоятельствах, поскольку это серьезно нарушит сложившуюся ментальную модель пользователей. В этом заключается общность чекбоксов и радиокнопок, теперь поговорим о различиях.

Главное различие заключается в том, что группа чекбоксов дают возможность пользователям выбрать любую комбинацию параметров, радиокнопки же позволяют выбрать только один параметр. Это сближает эти элементы со списками множественного и единственного выбора соответственно.

В группе не может быть меньше двух радиокнопок (как можно выбрать что-либо одно из чего-либо одного?).

У чекбокса есть три состояния (выбранное, не выбранное, смешанное), а у радиокнопки только два, поскольку смешанного состояния у неё быть просто не может (нельзя совместить взаимоисключающие параметры).

В группе радиокнопок как минимум одна радиокнопка должна быть проставлена по умолчанию.

Всякий раз, когда пользователю нужно предоставить выбор между несколькими параметрами, можно использовать либо чекбоксы, либо радиокнопки (или списки, но о них позже). Если параметров больше двух, выбор прост: если параметры можно комбинировать, нужно использовать чекбоксы (например, текст может быть одновременно *и* жирным *и* курсивным); если же параметры комбинировать нельзя, нужно использовать радиокнопки (например, текст может быть выровнен *или* по левому, *или* по правому краю).

Если же параметров всего два и при этом параметры невозможно комбинировать (т.е. либо ДА, либо НЕТ), решение более сложно.

И чекбоксы и радиокнопки крайне желательно расставлять по вертикали, поскольку это значительно ускоряет поиск нужного элемента

Внешний вид. Традиционно сложилось так, что чекбоксы выглядят как квадраты, а радиокнопки – как кружки. Нарушать это правило нельзя.

Желательно вертикально располагать чекбоксы и радиокнопки в группе, поскольку это облегчает поиск конкретного элемента.

Текст подписей. Каждая подпись должна однозначно показывать эффект от выбора соответствующего элемента. Поскольку как радиокнопки, так и чекбоксы, не вызывают немедленного действия, формулировать подписи к ним лучше всего

в форме существительных, хотя возможно использование глаголов (если изменяется не свойство данных, а запускается какое-либо действие). Подписи к стоящим параллельно кнопкам лучше стараться делать примерно одинаковой длины. Все подписи обязаны быть позитивными (т.е. не содержать отрицания). Повторять одни и те же слова, меняя только окончания подписей (например, «Показывать пробелы» и «Показывать табуляции»), в нескольких кнопках нельзя, в таких случаях лучше перенести повторяющееся слово в рамку группировки. Если подпись не помещается в одну строку, выравнивайте индикатор кнопки (кружок или квадрат) по первой строке подписи.

Вариант для панелей инструментов

Как чекбоксы, так и радиокнопки, бывают двух видов: описанные выше стандартные, и предназначенные для размещения на панелях инструментов.

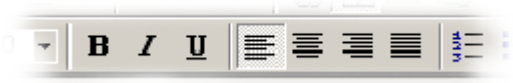


Рис. 35. Пример чекбоксов и радиокнопок на панели инструментов.

Справа расположены чекбоксы (шрифт может быть и жирным и курсивным), справа радиокнопки (абзац может быть выровнен либо по левому, либо по правому краю). Обратите внимание, что визуально чекбоксы и радиокнопки не различаются. У них есть определенный недостаток: они не различаются внешне

Списки

Все часто используемые списки функционально являются вариантами чекбоксов и радиокнопок.

Списки бывают пролистываемыми и раскрывающимися, причем пролистываемые могут обеспечивать как единственный (аналогично группе радиокнопок), так и множественный выбор (a la чекбокс); раскрывающиеся же работают исключительно как радиокнопки.

Но сначала необходимо рассказать об общих свойствах всех списков.

Ширина

Ширина списка как минимум должна быть достаточна для того, чтобы пользователь мог определить различия между элементами. В идеале, конечно, ширина всех элементов должна быть меньше ширины списка, но иногда это невозможно.

Пиктограммы.

Уже довольно давно в ПО нет технических проблем с выводом в списках пиктограмм отдельных элементов. Однако практически никто этого не делает. Это плохо, ведь пиктограммы обеспечивают существенное повышение субъективной привлекательности интерфейса и быстрее вызываются из ДВП, нежели слова.

Раскрывающиеся списки

Самым простым вариантом списка является раскрывающийся список. Помимо описанных выше родовых достоинств списков, раскрывающиеся списки

обладают одним существенным достоинством. Оно заключается в том, что малая высота списка позволяет с большой легкостью визуально отображать команды, собираемые из составляющих.

Рис. 36. Пример визуальной сборки команды из составляющих.

Данный метод значительно проще для понимания, нежели, например, ввод положительного значения для смещения вверх и отрицательного значения для смещения вниз без поддержки раскрывающимся списком.

Пролистываемые списки

Другим, более сложным вариантом списка является пролистываемый список. Пролистываемые списки могут позволять пользователям совершать как единственный, так и множественный выбор. Одно требование применимо к обоим типам списков, остальные применимы только к одному типу.

Размер. По-вертикали в список должно помещаться как минимум четыре строки, а лучше восемь. Напротив, список, по высоте больший, нежели высота входящих в него элементов, и соответственно, содержащий пустое место в конце, смотрится неряшливо. Требование выводить полоски прокрутки в больших списках кажется моветоном, но забывать о нем не следует.

Списки единственного выбора.

Список единственного выбора является промежуточным вариантом между группой радиокнопок и раскрывающимся списком. Он меньше группы радиокнопок с аналогичным числом элементов, но больше раскрывающегося списка. Соответственно, использовать его стоит только в условиях «ленивой экономии» пространства экрана.

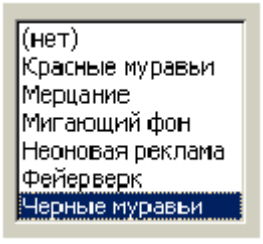


Рис. 37. Список единственного выбора.

Обратите внимание, что в ситуациях, когда все элементы помещаются в список без пролистывания, список работает в точности как группа радиокнопок.

Списки множественного выбора.

С точки зрения дизайна интерфейсов, списки множественного выбора интересны, прежде всего, тем, что их фактически нет в интернете. Технически создать список множественного выбора непроблематично, для этого в HTML есть даже специальный тег. Из-за такой убогой реализации списков браузерами,

использовать их, как правило, оказывается невозможно. Приходится использовать чекбоксы1.

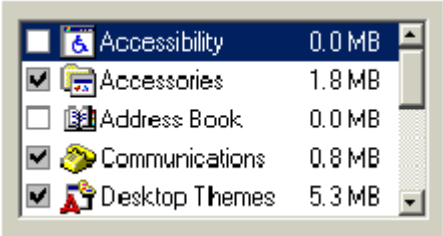


Рис. 38. Список множественного выбора с чекбоксами.

Гораздо лучше обстоят дела в ПО. Возможность безболезненно выводить в списке чекбоксы позволяет пользователям без труда пользоваться списками, а разработчикам – без труда эти списки создавать.

Комбобоксы

Комбобоксами (combo box), называются гибриды списка с полем ввода: пользователь может выбрать существующий элемент, либо ввести свой. Комбобоксы бывают двух видов: раскрывающиеся и расширенные. Оба типа имеют проблемы.

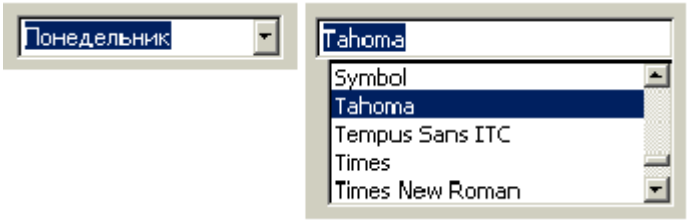


Рис. 39. Раскрывающийся комбобокс с установленным фокусом ввода (слева)
и расширенный комбобокс (справа).

У раскрывающегося комбобокса есть проблемы. Во-первых, такие комбобоксы выглядят в точности как раскрывающиеся списки, визуально отличаясь от них только наличием индикатора фокуса ввода (да и то, только тогда, когда элемент выделен). Это значит, что полноценно пользоваться ими могут только сравнительно продвинутые пользователи. В этом нет особой проблемы, поскольку комбобоксом все равно можно пользоваться, как обычным списком.

Поля ввода

Вместе с командными кнопками, чекбоксами и радиокнопками, поля ввода являются основой любого интерфейса. В результате требований к ним довольно много.

Размеры. Основная часть требований к полям ввода касается размера. Понятно, что размер по вертикали должен быть производным от размера вводимого текста – если текста много, нужно добавить несколько строк (нарушением этого правила регулярно грешат форумы, заставляющие пользователей вводить сообщения в поля ввода размером с ноготь).

С размерами по горизонтали интереснее. Конечно, ширина поля должна соответствовать объему вводимого текста, поскольку гораздо удобнее вводить

текст, который *видишь*. Менее очевидным является другое соображение: ширина поля ввода не должна быть больше объема вводимого в поле текста, поскольку частично заполненное поле выглядит как минимум неряшливо.

Ширина поля ввода не должна быть больше максимальной длины строки

Отдельной проблемой является ограничение вводимого текста. С одной стороны, ограничение хорошо для базы данных. С другой стороны, всегда найдутся пользователи, для которых поле ввода с ограничением вводимых символов окажется слишком маленьким. Поэтому этот вопрос нужно решать применительно к конкретной ситуации.

Код активации

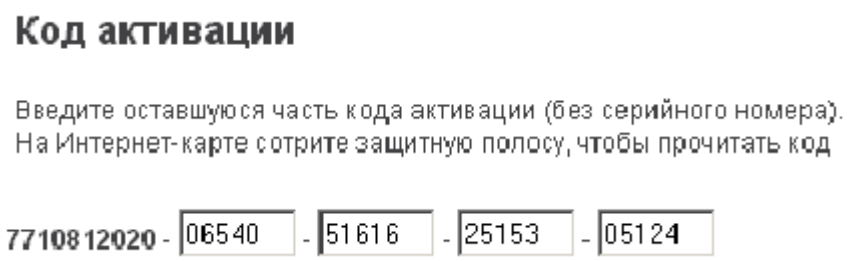


Рис. 40. Пример полей ввода, больших объема вводимых в них информации.

Мало того, что такие поля выглядят неряшливо, так они ещё и обманывают пользователей, показывая, что пользователь ввел не всю информацию. Вдобавок, после заполнения поля приходится самому перемещать фокус ввода, хотя с этим справилась бы и система.

Если же суммировать информацию из двух предыдущих абзацев, можно определить ***самую большую ошибку, которую разработчики допускают при создании полей ввода. Всякий раз, когда ширина поля ввода больше максимального объема вводимого в него текста, при этом объем вводимого текста ограничен, пользователи неприятно изумляются, обнаружив, что они не могут ввести текст, хотя место под него на экране имеется.***

Соответственно, делать поле ввода шире текста нельзя вообще.

Подписи.

Вопрос «где надо размещать подписи к полям ввода?» является одним из самых популярных среди программистов: битвы сторонников разных подходов, хоть и бескровны, но значительны. Аргументов и подходов тут множество.

Обычно действует следующее простое правило: в часто используемых экранах подписи должны быть сверху от поля (чтобы их было легче не читать), в редко же используемых подписи должны быть слева (чтобы всегда восприниматься и тем самым сокращать количество ошибок).

Подписи к полям ввода имеют определенное отличие от других подписей. В полях ввода подписи можно размещать не рядом с элементом, а внутри него, что позволяет экономить пространство экрана. Подпись при этом выводится в самом

поле ввода, точно так же, как и текст, который в него нужно вводить. Необходимо только отслеживать фокус ввода, чтобы при установке фокуса в поле убирать подпись. Это решение, будучи нестандартным, плохо работает в ПО, но неплохо работает в интернете.

Крутилки

Крутилка (spinner, little arrow) есть поле ввода, не такое универсальное, как обычное, поскольку не позволяет вводить текстовые данные1, но зато обладающее двумя полезными возможностями.

Рис. 41. Крутилка.

Во-первых, чтобы ввести значение в крутилку, пользователю не обязательно бросать мышь и переносить руку на клавиатуру (в отличие от обычного поля ввода). Поскольку перенос руки с место на место занимает сравнительно большое время (в среднем почти половину секунды и сбивает фокус внимания, отсутствие нужды в клавиатуре оказывается большим благом. Во всяком случае, случаи ввода значения в крутилку с клавиатуры достаточно редки, т.е. пользователи воспринимают крутилки целиком и полностью положительно.

Во-вторых, при вводе значения мышью система может позволить пользователям вводить только корректные данные, причем, что особенно ценно, в корректном формате. Это резко уменьшает вероятность человеческой ошибки. Таким образом, использование крутилок для ввода любых численных значений более чем оправдано.

Ползунки

Как и ранее описанные элементы управления, ползунки позволяют пользователям выбирать значение из списка, не позволяя вводить произвольное значение. Возникает резонный вопрос: зачем нужен ещё один элемент управления, если аналогичных элементов уже полно. Ответ прост: ползунки незаменимы, если пользователям надо дать возможность выбрать значение, стоящее в хорошо ранжирующемся ряду, если:

- ✓ значений в ряду много
- ✓ нужно передать пользователям ранжируемость значений.
- ✓ необходимо дать возможность пользователям быстро выбрать значение из большого их количества (в таких случаях ползунок оказывается самым эффективным элементом, хотя и опасен возможными человеческими ошибками).

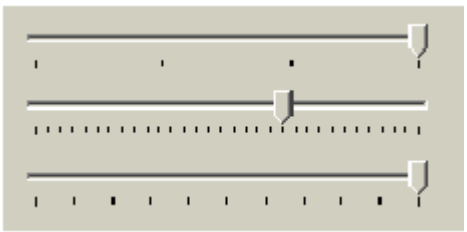


Рис. 42. Примеры ползунков.

Видно, что количество параметров в ползунке может быть весьма значительным (хотя расстояние между градациями не может быть меньше половины ширины ползунка, в случае необходимости вместить дополнительные значения можно просто увеличить ширину линейки).

Ползунки имеют интересный аспект. Их можно также использовать для выбора текстовых параметров, но только в случаях, когда эти параметры можно понятным образом отранжировать. Случаев таких немало, например, «завтрак», «обед» и «ужин», при отсутствии внешней связи ранжированию поддаются вполне.

Меню

При упоминании термина меню применительно к интерфейсу, большинство людей немедленно представляют стандартные раскрывающиеся меню. В действительности, понятие меню гораздо шире. Меню – это метод взаимодействие пользователя с системой, при котором пользователь выбирает из предложенных вариантов, а не предоставляет системе свою команду.

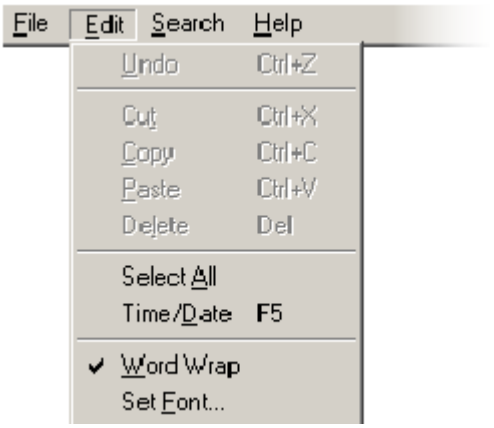


Рис. 43. Стандартное выпадающее меню.

Соответственно, диалоговое окно с несколькими кнопками (и без единого поля ввода) также является меню.

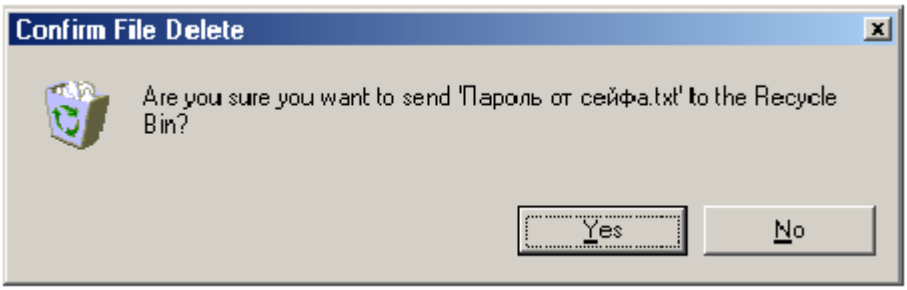


Рис. 44. Это тоже меню.

В настоящее время систем, которые не использовали бы меню в том или ином виде, практически не осталось. Объясняется это просто. Меню позволяет снизить нагрузку на мозги пользователей, поскольку для выбора команды не надо вспоминать, какая именно команда нужна и как именно её нужно использовать – вся (или почти вся) нужная информация уже содержится на экране. Вдобавок, поскольку меню ограничивает диапазон действий пользователей, появляется возможность в значительной мере изъять из этого диапазона ошибочные

действия. Более того: меню показывает пользователям объем действий, которые они могут совершить благодаря системе, и тем самым обучают пользователей (в одном из исследований было обнаружено даже, что меню является самым эффективным средством обучения1). Таким образом, в большинстве систем меню является объективным благом (они неэффективны, в основном, в системах с внешней средой или течением времени).

Типы меню

Существуют несколько различных таксономий меню, но основной интерес представляют только две из них. Первая таксономия делит меню на два типа:

- ✓ Статические меню, т.е. меню, постоянно присутствующие на экране. Характерным примером такого типа меню является панель инструментов.
- ✓ Динамические меню, в которых пользователь должен вызвать меню, чтобы выбрать какой-либо элемент. Примером является обычное контекстное меню.

В некоторых ситуациях эти два типа меню могут сливаться в один:

например, меню, состоящее из кнопок доступа к меню (см. стр. 68), могут работать и как статические (пользователи нажимают на кнопки) и как динамические (пользователи вызывают меню).

Вторая таксономия также делит меню на два типа:

- ✓ Меню, разворачивающиеся в пространстве (например, обычное выпадающее меню). Всякий раз, когда пользователь выбирает элемент нижнего уровня, верхние элементы остаются видимыми.
- ✓ Меню, разворачивающееся во времени. При использовании таких меню элементы верхнего уровня (или, понимая шире, уже пройденные элементы) по тем или иным причинам исчезают с экрана. Например, в предыдущей иллюстрации диалоговое окно с меню перекрыло элемент управления, которым это меню было вызвано.

Каждый тип меню в обеих таксономиях имеет определенные недостатки.

Статические меню из первой таксономии, как правило, обеспечивают меньшую скорость работы, лучше обучают пользователей, но зато занимают место на экране. Напротив, с динамическими меню ситуация обратная. Во второй таксономии первый тип (меню, разворачивающиеся в пространстве) обеспечивает большую поддержку контекста действий пользователей, но эта поддержка обходится в потерю экранного пространства. Второй тип более бережно использует пространство, но зато хуже поддерживает контекст.

Реальность, впрочем, оказывается несколько шире обеих таксономий. Например, мастер (см. «Последовательные окна» на стр. 99), являясь и динамическим меню из первой таксономии, и разворачивающимся во времени меню из второй, не оказывается более быстрым, чем, например, раскрывающееся меню. Но объем и специфика входящих в него элементов управления не позволяют, как правило, сделать из него какое-либо другое меню, например, раскрывающееся.

Поэтому очень полезно научиться анализировать влияние и взаимопроникновение разных типов меню, а также осознавать их место в интерфейсе. Например, контекстное меню на ином уровне абстракции оказывается временным (т.е. динамическим) диалоговым окном, только с нестандартной структурой. Понимание этой структуры позволяет определить, какие элементы управления, помимо кнопок, можно использовать в таком меню, чтобы оно обрело как достоинства меню, так и достоинства диалогового окна. К сожалению, объем этой книги не позволяет более полно описать эту тему. Поэтому в этом разделе будут описаны только главные и контекстные меню.

Устройство меню

На эффективность меню наибольшее влияние оказывают устройство отдельных элементов и их группировка. Несколько менее важны другие факторы, такие как выделение элементов и стандартность меню.

Устройство отдельных элементов

Самым важным свойством хорошего элемента меню является его название. Название должно быть самым эффективным из возможного. В отличие от кнопок в диалоговых окнах, элементы главного меню практически никогда не несут на себе контекста действий пользователя, просто потому, что в любой момент времени доступны все элементы. Это значит, что к наименованию элементов меню нужно подходить весьма тщательно, тщательней, нежели ко всему остальному. Впрочем, помимо тщательности (и таланта, к слову говоря) нужно ещё кое-что. Обязательно нужно убедиться, что выбранное название понятно целевой аудитории. Сделать это просто – пользователю нужно сообщить название элемента и попросить его сказать, что этот элемент меню делает.

Нелишне заметить, что функциональность, не отраженная названием элемента, с большой степенью вероятности не будет найдена значительной частью аудитории. Поэтому не стоит умещать в диалоговое окно какую-либо функцию, если её существование в этом окне невозможно предсказать, глядя на соответствующий элемент меню.

Не делайте элементов меню, часть функциональности которых не влезает в текст элемента

Особо стоит остановиться на склонении текста. В отличие от диалоговых окон, в которых кнопки прямого и отложенного действия выглядят и действуют по-разному, в меню нет четкой разницы между этими элементами. Единственным способом разграничения этих элементов является текст, так что нужно очень тщательно подходить к тому, чтобы элементы, запускающие действия, были глаголами в форме инфинитива (как командные кнопки). Впрочем, часто глагол приходится выкидывать вообще, чтобы переместить значимое слово ближе в начало текст элемента. Нужно это, чтобы повысить скорость распознавания. Повысить её можно всего одним способом: главное (т.е. наиболее значимое) слово в элементе должно стоять в элементе первым. Обратите внимание, что короткий текст элемента, без сомнения, быстро читаясь, совершенно

необязательно быстро распознается. Поэтому не стоит безудержно сокращать текст элемента: выкидывать нужно все лишнее, но не более.

Пиктограммы в меню

Пиктограммы в меню, если они повторяют пиктограммы в панели инструментов, обладают замечательной способностью обучать пользователей возможностям панели. Помимо этого они здорово ускоряют поиск известного элемента и точность его выбора, равно как и общую разборчивость меню. Таким образом, пиктограммы в меню объективно хороши (только стоят дорого, к сожалению). Это очевидный факт. Теперь менее очевидный: пиктограммы лучше работают, когда ими снабжены не все элементы. Когда все элементы имеют пиктограммы, разборчивость каждого отдельного элемента падает: в конце концов, пиктограммы всех ненужных в данное время элементов являются визуальным шумом. Когда же пиктограммами снабжены только самые важные элементы, их разборчивость повышается (а разборчивость остальных не понижается), при этом пользователям удается легче запоминать координаты элементов («элемент сразу под второй пиктограммой»).

Не снабжайте пиктограммами все элементы меню, снабжайте только самые важные

Переключаемые элементы.

Особого внимания заслуживают случаи, когда меню переключает какие-либо взаимоисключающие параметры, например, показывать или не показывать палитру. Тут есть несколько возможных способов. Можно поместить перед переключателем галочку, показывая, что он включен (если же элемент снабжен пиктограммой, можно её утапливать). Заранее скажу, что это лучший метод. Можно не помещать галочку, зато инвертировать текст элемента: например, элемент Показывать сетку превращается в Не показывать сетку. Это плохо по многим причинам. Во-первых, в интерфейсе желательно не употреблять ничего негативного: в меньшей степени потому, что негативность слегка снижает субъективное удовлетворение; в большей степени потому, что она снижает скорость распознавания текста (главное слово не первое, нужно совершить работу, чтобы из отрицания вычислить утверждение). Во-вторых, если изъять «не» и переформулировать одно из состояний элемента, пользователям будет труднее осознать, что два разных элемента на самом деле есть один элемент. Таким образом, галочка предпочтительнее. Всегда формулируйте текст в интерфейсе без использования отрицаний

Предсказуемость действия.

Пользователей нужно снабжать чувством контроля над системой. Применительно к меню это значит, что по виду элемента пользователи должны догадываться, что произойдет после выбора. Сделать это неимоверно трудно, поскольку на экране нет места под такие подсказки. Можно сделать только одно, но сделать это нужно обязательно: нужно показать пользователям, какой элемент запускает действие или меняет параметр, а какой открывает окно с продолжением

диалога. Почти во всех ОС стандартным индикатором продолжения диалога является многоточие после текста элемента, так что пользоваться этим признаком стоит везде, включая интернет. Также необходимо показывать, какой элемент срабатывает сразу, а какой открывает элементы меню нижнего уровня (в любой ОС это делается автоматически, в интернете нужно не забывать делать это вручную).

Это же правило касается и гипертекстовых ссылок вообще (они тоже меню). Пользователи испытывают значительно большее чувство контроля, когда имеют возможность предсказать, куда их ссылка приведет (при этом снижается количество ошибочных переходов). Таким образом, нестандартные ссылки (т.е. ссылки на другой сайт, на почтовый адрес, на файл, на узел FTP, на долго загружающуюся страницу и т.д.) полезно снабжать характерными для них признаками, например, ссылку на почтовый адрес пиктограммой письма.

Группировка элементов

Второй составляющей качества меню является группировка его элементов. В большинстве меню группировка оказывает не меньшее значение при поиске нужного элемента, нежели само название элемента, просто потому, что даже идеальное название не сработает, если элемент просто нельзя найти.

Чтобы уметь эффективно группировать элементы в меню, нужно знать ответы на три вопроса: зачем элементы в меню нужно группировать, как группировать элементы и как разделять группы между собой.

Зачем элементы в меню нужно группировать.

Меню, группы элементов в котором разделены, сканируется значительно быстрее обычного, поскольку в таком меню больше «точек привязки» (точно также как и в меню с пиктограммами). К тому же наличие явных разделителей многократно облегчает построение ментальной модели, поскольку не приходится гадать, как связаны между собой элементы. Наконец, в объемных меню группировка элементов облегчает создание кластеров в кратковременной памяти, благодаря чему всё меню удается пометить в кратковременную память.

Как группировать элементы.

Каждый знает, или, во всяком случае, догадывается, что элементы в меню нужно группировать максимально логично. Поспорить с этим утверждением нельзя, но от этого его проблематичность не уменьшается.

Взаимоисключающие элементы желательно помещать в отдельный уровень иерархии

Дело в том, что существует множество типов логики. Есть логика разработчика, который знает все функции системы. Есть логика пользователя, который знает только меньшую часть. При этом практика показывает, что эти типы логики в значительной мере не совпадают. Поскольку пользователи важнее, нужно сгруппировать меню в соответствии с их логикой.

Для этого используется очень простой и надежный метод, называемый карточной сортировкой (см. стр. 120).

Как разделять группы между собой.

Существует два основных способа разделять группы: между группами можно помещать пустой элемент (разделитель) или же размещать отдельные группы в разных уровнях иерархии.

Второй способ создает более четкое разделение: в меню Файл, например все элементы более близки друг другу (несмотря на разделители), чем элементы других меню. В то же время выбор конкретного способа диктуется результатами карточной сортировки, так что интерес представляет только вопрос «как должны выглядеть и действовать разделители».

Для разграничения групп традиционно используют полоски. Это надежное, простое решение, другой разговор, что с дизайнерской точки зрения полоски плохи, поскольку представляют собой визуальный шум. Гораздо правильнее, но и труднее, использовать только визуальные паузы между группами, как это сделано, например, в MacOS X.

Глубина меню.

Наличие многих уровней вложенности в меню приводит к там называемым «каскадным ошибкам»: выбор неправильного элемента верхнего уровня неизбежно приводит к тому, что все следующие элементы также выбираются неправильно. При этом широкие меню больше нравятся пользователям. Поэтому большинство разработчиков интерфейсов стараются создавать широкие, а не глубокие меню1.

К сожалению, у широких меню есть недостаток: они занимают много места. Это значит, что, начиная с определенного количества элементов, меню физически не сможет оставаться широким, оно начнет расти в глубину. Возникает проблема, которую надо решать. Итак, проблема заключается в том, что велика вероятность каскадных ошибок. Чтобы снизить их число, нужно повысить вероятность того, что пользователи будут правильно выбирать элементы верхних уровней. Чтобы повысить эту вероятность, нужно заранее снабдить пользователей контекстом.

При перемещении по меню пользователь действует по определенному алгоритму:

1 Выбирая элемент первого уровня, он выбирает элемент, «нужность» которого кажется ему максимальной.

2 После выбора он видит список элементов второго уровня, при этом он оценивает вероятность соответствия всех элементов второго уровня его задаче и одновременно выбирает наиболее вероятный элемент. При этом в уме он держит контекст, т.е. название элемента первого уровня.

3 Если ни один из элементов не кажется пользователю достаточно вероятным, пользователь возвращается на первый уровень.

4 Если какой-то элемент удовлетворяет пользователя, он выбирает его и получает список элементов третьего уровня. Действия из второго и третьего шагов повторяются применительно к новым элементам меню.

Видно, что действия пользователя при поиске нужного элемента отчетливо цикличны, при этом на каждом шаге есть вероятность ошибок. При этом с каждым новым уровнем меню объем контекста, который приходится держать в голове,

непрерывно возрастает. При этом, если пользователь всё-таки не находит нужного элемента, весь этот контекст оказывается ненужным. Хранение же контекста, даже не засчитывая усилия, затрачиваемые на выбор элемента, есть довольно существенная работа. Её объем лучше уменьшить.

Теперь рассмотрим другой вариант: пользователь по самому элементу может предугадать его содержимое, т.е. при поиске элемента в меню не столько оценивает контекст, сколько просто ищет нужный элемент. Эта возможность есть в любом случае, поскольку элемент имеет хоть сколько-нибудь значимый идентификатор (т.е. его название). Но она, как правило, довольно слаба и почти всегда допускает неоднозначность. Усилить её можно наличием аннотации к каждому элементу, но эту аннотацию никто не будет читать.

Есть другой метод, и этот метод есть лучшее, что дал интернет науке о проектировании интерфейсов: в качестве аннотации к элементу можно показывать наиболее популярные элементы следующего уровня.

Контекстные меню

Преимущество контекстных (всплывающих) меню заключается в том, что они полностью встраиваются в контекст действий пользователей: не нужно переводить взгляд и курсор в другую область экрана, практически не нужно прерывать текущее действие для выбора команды. При этом они не занимают места на экране, что всегда ценно. С другой стороны, из-за того, что они не находятся всё время на экране, они практически неспособны чему-либо научить пользователя.

Не делайте контекстные меню единственным способом вызова какой*либо функции

Поскольку основной причиной появления контекстных меню является стремление максимально повысить скорость работы пользователей, на их размер и степень иерархичности накладываются определенные ограничения. Если меню будет длинным, пользователям придется сравнительно долго возвращать курсор на прежнее место, так что привлекательность нижних элементов окажется под вопросом. Поэтому лучше сокращать размер контекстных меню до разумного минимума (порядка семи элементов).

К тому же не надо забывать, что главное меню *не всегда* перекрывает выделенный (т.е. актуальный объект), а контекстное меню – *почти всегда* (как-никак оно вызывается на самом объекте). В большинстве же случаев перекрытие актуального объекта нежелательно (сбивается контекст). Мы не можем сделать в этой ситуации ничего, кроме как уменьшить размер меню, в расчете, что маленькое меню будет перекрывать малое количество информации. Разумеется, если точно известно, что оперируемый объект совсем уж мал, сокращать объем меню бесполезно. Другая особенность контекстных меню – иерархия. В обычном меню иерархия имеет хотя бы одно достоинство: при обучении она позволяет упорядочивать элементы меню и тем самым делать его понятнее. В контекстных же меню обучающая функция не играет никакой роли, поскольку такими меню пользуются только опытные пользователи. Иерархия элементов теряет свое единственное достоинство, не теряя ни одного недостатка. Поэтому делать

иерархические контекстные меню можно, ничего плохого в этом нет, но необходимо сознавать, что вложенными элементами почти никто не будет пользоваться (тем более что вложенность сбивает контекст действий).

Система сначала должна показывать максимально релевантную информацию, затем всё остальное

Последнее отличие контекстных меню от обычных заключается в том, что в них очень важен порядок следования элементов. В главном меню не обязательно стремиться к тому, чтобы наиболее часто используемые элементы были самим первыми – все равно курсор придется возвращать к рабочему объекту, так что разницы в дистанции перемещения курсора практически нет. В контекстном же меню ситуация обратная – чем дальше нужный элемент от верха меню, тем больше придется двигать курсор.

Поэтому правило релевантности в таких меню действует в полной мере.

Окна

Поскольку разработка интерфейса заключается в основном в том, чтобы правильно помещать правильные элементы управления в правильные диалоговые окна или экраны, окна требуют не меньше заботы, чем элементы управления.

Типы окон

Современная наука знает несколько типов окон, а именно:

_ главные окна программы

_ окна документа

_ режимные диалоговые окна (о разнице между режимными и безрежимными окнами

_ безрежимные диалоговые окна

_ палитры

_ окна браузера (поскольку используемая в интернете технология существенно отличается от технологии ПО, этот тип окон стоит несколько особняком).

При этом доля отдельных типов в общем пироге со временем изменяется: окна документов, как будет показано ниже, отмирают, заменяясь окнами программ, режимные диалоговые окна сменяются безрежимными, а безрежимные, в свою очередь, палитрами. Интересно, что идея палитр тоже клонится к закату (палитры сменяются панелями инструментов, причины этого опять-таки рассмотрены ниже), так что в будущем, скорее всего, в ПО останутся только окна программ, панели инструментов и безрежимные диалоговые окна (которые разработчики поленятся переделывать). Но об этом отдельно.

Вопросы к экзамену

1. Общее представление об информационной системе
2. Классификация интерфейсов
3. Пакетная технология

4. Технология командной строки.
5. Графический интерфейс
6. Простой графический интерфейс.
7. WIMP - интерфейс
8. Речевая технология
9. Биометрическая технология ("Мимический интерфейс".)
10. Семантический (Общественный) интерфейс.
11. Типы пользовательских интерфейсов и этапы их разработки.
12. Типы интерфейсов.
13. Психофизические особенности человека, связанные с восприятием, запоминанием и обработкой информации
14. Пользовательская и программная модели интерфейса
15. Классификации диалогов и общие принципы их разработки
16. Факторы оценки пользовательских интерфейсов
17. Скорость выполнения работы
18. Правила GOMS
19. Длительность интеллектуальной работы
20. Непосредственное манипулирование
21. Потеря фокуса внимания
22. Длительность физических действий
23. Длительность реакции системы
24. Человеческие ошибки
25. Типы ошибок
26. Блокировка потенциально опасных действий до получения подтверждения
27. Проверка действий пользователя перед их принятием
28. Самостоятельный выбор команд
29. Два уровня ошибок и обратная связь
30. Обучение работе с системой
31. Почему пользователи учатся
32. Средства обучения
33. Понятность системы
34. Ментальная модель
35. Метафора
36. Аффорданс.
37. Стандарт
38. Обучающие материалы
39. Сообщения об ошибках.
40. Спиральность
41. Субъективное удовлетворение
42. Эстетика
43. Каким должно быть сообщение об ошибке
44. Различные элементы управления
45. Кнопки
46. Командные кнопки
47. Кнопки доступа к меню

48. Чекбоксы и радиокнопки
49. Вариант для панелей инструментов
50. Списки
51. Раскрывающиеся списки
52. Пролистываемые списки
53. Списки единственного выбора.
54. Списки множественного выбора.
55. Комбобоксы
56. Поля ввода
57. Код активации
58. Подписи.
59. Крутилки
60. Ползунки
61. Меню
62. Типы меню
63. Устройство меню
64. Устройство отдельных элементов
65. Пиктограммы в меню
66. Переключаемые элементы.
67. Предсказуемость действия.
68. Группировка элементов
69. Глубина меню.
70. Контекстные меню
71. Окна
72. Типы окон