дизайн пользовательского интерфейса V1.2

влад. в. головач

usethics

Затаив дыхание, Пончик поднялся по лестничке и нажал кнопку у двери. Дверь отворилась. Пончик вылез из пищевого отсека и принялся бродить по кривому коридорчику, стараясь отыскать дверцу лифта. Он не был так хорошо знаком с устройством ракеты, как Незнайка, поэтому несколько раз обошел коридорчик вокруг, каждый раз попадая к пищевому отсеку. Опасаясь, что Незнайка проснется и обнаружит его исчезновение, Пончик снова стал нервничать и терять соображение. Наконец ему все же удалось отыскать дверцу лифта. Недолго думая он забрался в кабину и нажал первую попавшуюся кнопку. Кабина, вместо того чтобы опуститься вниз, поднялась вверх. Но Пончик не обратил на это внимания и, выйдя из кабины, принялся искать дверь шлюзовой камеры, через которую можно было выйти наружу.
В шлюзовую камеру он, конечно, попасть не мог, потому что ее здесь не было, а попал вместо этого в кнопочную кабину и стал ощупывать в темноте стены, стараясь найти выключатель. Выключателя ему не удалось обнаружить, но посреди кабины он наткнулся на небольшой столик, на котором нащупал кнопку. Вообразив, что посредством этой кнопки включается свет, Пончик нажал ее и сразу подскочил кверху, оказавшись в состоянии невесомости. Одновременно с этим он услышал мерный шум заработавшего реактивного двигателя.

Некоторые самые догадливые читатели, наверно, сразу сообразили, что Пончик нажал как раз ту кнопку, которая включала электронную управляющую машину. А электронная управляющая машина, как это и было предусмотрено конструкторами, сама собой включила прибор невесомости, реактивный двигатель и все остальное оборудование, благодаря чему ракета отправилась в космический полет в тот момент, когда этого никто не ожидал.

Николай Носов. Незнайка на Луне

Моим родителям.

Оглавление

Введение

В нашей стране происходят два независимых процесса, делающие дизайн интерфейсов не профессией отдельных людей, но частью деятельности многих.

Во-первых, зарождается нормально устроенная индустрия заказной разработки программного обеспечения. Конечно, этот бизнес существовал и раньше, но успех на нем во многом определялся не качеством работы, но школьным знакомством с заказчиком. Сейчас ситуация изменилась. Внутренний рынок насыщен и стоимость работы на нем низка. Приходится, по примеру Индии, работать на западный рынок. Требования же к качеству работы там гораздо выше отечественных, причем в эти требования входит качество интерфейса.

Вторым процессом является развитие интернета и соответствующая разработка сайтов. Спрос родил предложение, и в стране образовалось большое количество профессиональных веб-дизайнеров, совершенно не знающих принципы конструирования интерфейсов. Рынок же веб-дизайна обнаруживает те же тенденции, что и рынок программирования (т. е. такая же ориентация на запад, примерно такие же требования к качеству). Таким образом, образовалось две категории людей (программисты и веб-дизайнеры), профессиональный успех которых напрямую связан с качеством создаваемого ими пользовательского интерфейса.

Но эта книга будет полезна не только программистам и веб-дизайнерам. Дизайн пользовательского интерфейса вовсе не дитя компьютера. Фактически, одни и те же принципы применимы как при проектировании программы, так и при проектировании тостера (различаться, скорее, будут конкретные методы реализации интерфейса). Это значит, что книга может оказаться полезной также и промышленным дизайнерам.

Об этой книге и обо мне

Когда я только начал писать эту книгу, я твердо решил избегать в ней стиля «Десять советов начинающему акушеру», к сожалению, крайне популярному в литературе сходной тематики. Стиль этот, характерный, прежде всего, неумеренным употреблением «эвристик»[1] типа «Избегайте бить пользователей палкой по голове», будучи, без сомнения, формально правильным, имеет два существенных недостатка – он так же интересен, как таблица умножения (а) и не вызывает никакого желания следовать этим эвристикам (б). Как задумал, так и получилось (хотя эвристик, как таковых, я вовсе не избегал).

Далее. Эта книга не могла бы родиться, если бы я в течение нескольких лет не работал над своим авторским проектом, посвященным, в том числе, и дизайну интерфейса (сфера моих интересов довольно-таки широка). В частности, именно на сайте я обкатал свой стиль, отличающийся непомерной категоричностью суждений. Если я считаю какое-либо действие необходимым, я стараюсь писать так, как будто нет ни технических трудностей реализации, ни сложившихся стандартов, ни чего-либо ещё. Многие читатели сайта жаловались на мой стиль (почему я и пишу об этом здесь). Но я совершенно уверен, что мой подход правилен. Эта книга называется «Дизайн пользовательского интерфейса», а дизайн потому и дизайн, что не терпит косности в сознании. К нему надо подходить творчески. Если бы я хотел написать книгу, о том, как делают *все*, эта книга называлась бы «Проектирование пользовательского интерфейса». Но я не хотел. Человеко-компьютерное взаимодействие, как проблематика, очень молода, так что у нас есть хорошие шансы изобрести что-нибудь совершенно новое и прогрессивное. Именно это сделает меня счастливым. Таким образом, я пытался создать книгу, по возможности не отвечающую на вопрос *Как*, но посвященную ответам на вопросы *Зачем* и *Почему*. Я старался оставить читателю удовольствие самому сделать нужные выводы.

1. Эвристика [гр. heuresko – отыскиваю, открываю] – специальные методы решения задач (эвристические методы), которые обычно противопоставляются формальным методам решения, опирающимся на точные математические модели. Использование эвристических методов (эвристик) сокращает время решения задачи по сравнению с методом полного ненаправленного перебора возможных альтернатив; получаемые решения не являются, как правило, наилучшими, а относятся лишь к множеству допустимых решений; применение эвристических методов не всегда обеспечивает достижение поставленной цели. Большая Российская энциклопедия

И, наконец, самое главное. Эта книга почти полностью основывается на *моем личном* опыте, а не анализе и компиляции чужих работ по человеко-компьютерному взаимодействию, когнитивной и инженерной психологии, а также эргономике. Все идеи, эвристики и аксиомы имеют источником мои собственные наблюдения (за исключением немногих случаев, когда я явно ссылался в тексте на работы других авторов). Вероятно, текст этой книги во многом копирует чужие работы, но это происходит только потому, что я и авторы этих работ независимо пришли к одинаковым результатам.

Чего здесь нет

Когда я написал две первых страницы, я понял, что написание всеобъемлющего труда нереально: во-первых, это требует количества времени, которым я не располагаю, а во-вторых, что более важно, это никто не сможет читать. Более того. Оказалось, что даже более-менее систематическое описание тоже невозможно, поскольку это потребует написания такого количества словесной руды, что пробраться через неё обычному читателю будет чрезвычайно тяжело. Осознав это, я изменил план книги так, чтобы она состояла из трех частей:

- **Факторы.** Эта часть, фактически состоящая из отдельных, не слишком связанных между собой эссе, описывает основные аспекты дизайна интерфейсов.
- **Инвентарь.** В этой части описываются основные «кирпичи», из которых состоит любой графический интерфейс, начиная кнопками и заканчивая окнами.
- **Процесс.** Часть, описывающая сам процесс дизайна интерфейса.

И ещё раз более того. Нежелание писать «Библию проектирования интерфейса» привело к тому, что я почти не писал о низкоуровневом интерфейсе[1] (что важно для разработчиков игр и флаш-роликов), равно как и о проектировании интерфейсов управления динамическими системами (что важно разработчикам того, что раньше называлось АСУТП). Разумеется, эта книга будет полезна и для этих людей, но не так полезна, как могла бы быть.

Терминология

Как это ни печально, но русская терминология интерфейса фактически отсутствует. Сомневающиеся в этом факте могут убедиться в моей правоте, узнав, как в русском глоссарий фирмы Microsoft, которая, по понятным причинам, является законодателем мод в этой области, переводятся термины «radio button» и «checkbox». В английском языке тоже не всё гладко, так, например, элемент управления, называемый в мире Microsoft «spinner», в мире Apple называется «little arrows». Но буржуям, конечно, гораздо легче.

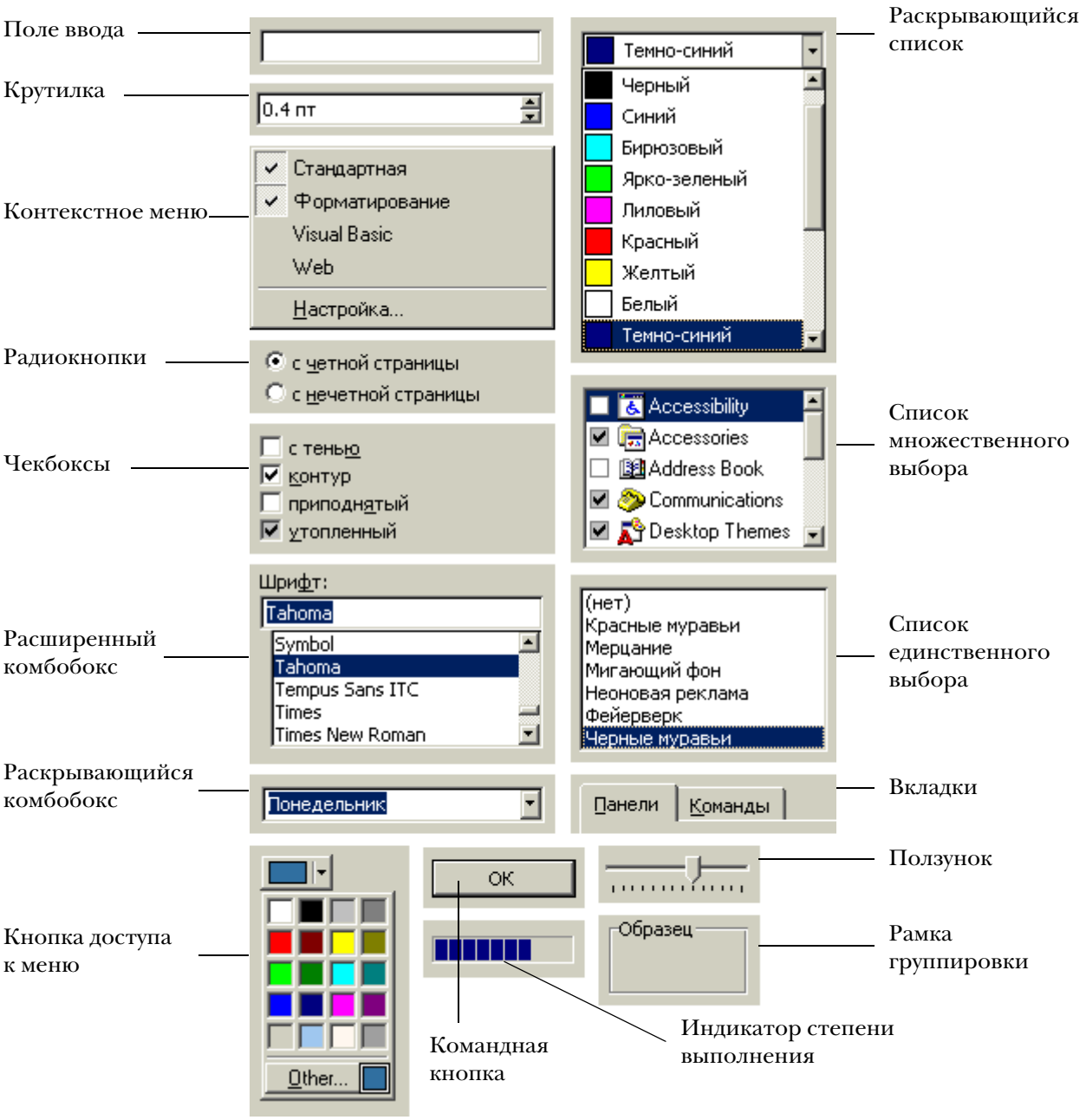
Так или иначе, но при написании этой книги передо мной встала нешуточная проблема: как называть элементы управления? Обнаружилось (в который раз), что не для всех элементов можно подобрать формально правильные названия, т. е. такие названия, которые как-то связаны с обозначаемым элементом. У тех же элементов, которым такие названия подобрать можно, эти названия получаются длинными, «как “Фауст” Гёте». Так или иначе, получалось плохо, даже массовый опрос коллег не помог. Таким образом, у меня была альтернатива: я мог воспользоваться глоссарием Microsoft (во многих отношениях плохоньким), мог перевести сам нужные термины (но они получались непроизносимыми) или мог оставить все термины по-английски (или, как вариант, перевести их транслитерацией). Я выбрал срединный путь.

Элементы управления я буду называть их английским названием в транслитерации, причем, если возможен правильный, но труднопроизносимый вариант, я буду писать его в скобках после первого появления транслитерированного термина. Есть, однако, названия, которые переводятся и хорошо, и сравнительно кратко. Их я буду переводить, стараясь делать это «по-майкрософтовски».

Аргументирую я свое решение следующим образом. Письменный язык должен совпадать с устным. Представить себе программиста, который говорит другому программисту «а не вставить ли тебе сюда парочку переключателей единственного выбора?» совершенно невозможно[2]. Значит, в данном конкретном случае транслитерация имеет право на существование. Всё равно все говорят «чекбокс». Вот и пусть говорят дальше.

1. В отличии от высокоуровневого интерфейса, в котором единицами величины являются кнопки, меню, окна и т.д., в низкоуровневом интерфейсе единицами являются пиксели и миллисекунды (например, «через сколько миллисекунд нужно принимать новое нажатие клавиши?» или «на сколько пикселей можно сместить курсор во время двойного щелчка, чтобы щелчок остался двойным?»).

2. Этот пример напоминает известный анекдот, в котором один сварщик говорит другому: «скажи же, о мой друг и коллега Иван, зачем ты капаешь мне за шиворот нагретый до семисот градусов Цельсия припой?».



Для простоты я собрал все элементы управления, о которых идет речь в книге, в одну таблицу, так что, встретив непонятное название, вы можете посмотреть, какому элементу оно соответствует.

Благодарности

Эта книга никогда бы не появилась на свет (или бы появилась, но значительно худшая), если бы не следующие люди:

1 Александр Белышкин, Ярослав Перевалов, Андрей Седельников и Анна Волошинская, которые прочли рукопись и высказали мне огромное количество на удивление конструктивных замечаний.
2 Мой отец, который сначала научил меня самостоятельно мыслить, потом заставил писать, а потом тоже прочел рукопись и высказал.
3 Моя жена Катя, долгие месяцы терпевшая мужа-публициста, и ни разу ему не высказавшая.
4 Мои друзья, которые высказывали мне очень многое, но не с желанием обидеть, а просто по дурости.

Появление новой версии стало возможно благодаря множеству людей, которые нашли в книге множество ошибок – начиная от фактологических и кончая опечатками. Они сообщили об этих ошибках мне и поступили правильно, ибо книга стала гораздо лучше. Этих людей очень много, я боюсь кого-нибудь пропустить, поэтому никого — увы — не называю поименно.

Большое всем спасибо и да пребудет с вами благословение Аллаха.

Часть i

Факторы

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

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 (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 (см. «Предсказание скорости» на стр. 120).

Длительность интеллектуальной работы

Согласно Дональду Норману[1], взаимодействие пользователя с системой (не только компьютерной) состоит из семи шагов:

1. формирование цели действий
2. определение общей направленности действий
3. определение конкретных действий
4. выполнение действий
5. восприятие нового состояния системы
6. интерпретация состояния системы
7. оценка результата.

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

К сожалению, существенно повысить скорость собственно мышления пользователей невозможно. Тем не менее, уменьшить влияние факторов, усложняющих (и, соответственно, замедляющих) процесс мышления, вполне возможно. 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

1. D. Norman, «The Design of Everyday Things», Doubleday (1988).

> 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 кипяток 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:

1. Возможно, что когда (и если) появится нормально работающее голосовое управление, метод командования плитой («и мочалок командир») станет приемлемым. В настоящее время, однако, уровень технологий таков, что от пользователя скорее требуется выбрать в меню команду «Кухня > Плита > Газ > Зажечь».

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, просмотрев 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. 76). Но поскольку это достоинство не имеет прямого отношения к скорости работы, можно смело сказать, что меню из состязания выбыло.

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 (см. стр. 100). 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 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. 40). А теперь предположим, что пользователь собрался стереть важный файл, который стирать не рекомендуется. Ни один из 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. 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. 47), 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» интер-

фейса 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 M». 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.

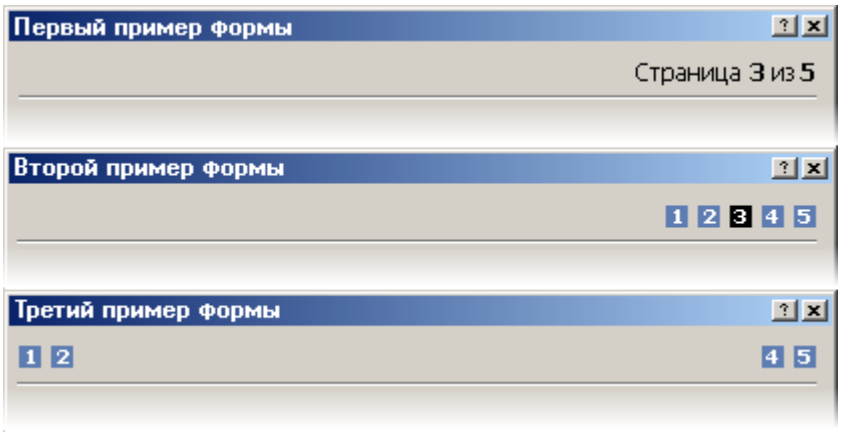


Рис. 1. Три варианта индикации степени заполнения экранной формы. Второй и третий варианты значительно визуальнее. Не используйте в подобных случаях ползунки (см. стр. 75).

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, в наиболее практичной формулировке ставшее известным как Закон Фитса:

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

1. P. Fitts, «The informational Capacity of the human motor system in controlling amplitude of movement», Journal of Experimental Psychology, 47, (1954), 381-391.

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 – она расположена впритык к краю экрана, но кнопки отделены от края экрана тремя пустыми пикселями. И скорость работы снижается, и место теряется. © Microsoft.

Нулевая дистанция до кнопки. Рассмотрим контекстное меню, вызываемое по нажатию правой кнопки мыши. Оно всегда открывается под курсором, соответственно расстояние до любого его элемента всегда минимально[1]. 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 лучше[2], 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 KLM, см. «Предсказание скорости» на стр. 120), 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. 97.

1. Еще лучше было бы делать контекстные меню не вертикальными, но круглыми, когда элементы расположены вокруг курсора. К сожалению, на них практически невозможно писать текст элементов, так что распространения они не получили.
2. Есть еще и альтернативное решение. Во многих мышиных драйверах есть возможность автоматически перемещать курсор ко всем появившимся на экране кнопкам, выбранным по умолчанию. Это увеличивает скорость работы, но зато и увеличивает вероятность человеческих ошибок, поскольку пользователь может, не разобравшись, нажать не на ту кнопку.

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, для начала следующего действия в их трудовом процессе им нужна распечатка, которой ещё нет.

Проблема в том, что сразу после того, как человек отвлекается, системе зачастую 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 полезно пользовать всегда, когда невозможен первый метод (разумеется, за исключением случаяев, когда ответ *Да* может иметь катастрофические последствия).

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

Есть и другая причина отвлечения пользователя. Пользователь запускает какой-либо процесс. Система показывает ему индикатор степени выполнения. Процент выполнения за минуту едва доходит до четверти размера индикатора. Пользователь экстраполирует эти данные и резонно решает, что у него есть три минуты, чтобы размяться. Однако, как только он отходит от компьютера, процент выполнения с нечеловеческой скоростью начинает расти и за секунду доходит до максимума. Процесс успешно заканчивается, а пользовать еще три минуты бездельничает[1].

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 ско-

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

ростью, 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.

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). Классический сюжет из жизни летчика, который после взлета хотел убрать шасси, но вместо этого включил систему аварийного катапультирования, возник отнюдь не на пустом месте.

Существование несуществующего

Эта глава о человеческих ошибках. Поэтому вначале необходимо сказать главное: они не существуют.

Как же, воскликнете вы, как же не существуют? Ведь человек при работе с компьютером постоянно совершает ошибки, сами мы, например, делали их не раз и не два!

Очень просто, отвечу я. Дело в том, что 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.

Попробуйте вслушаться в любой разговор между людьми. Вы обнаружите, что он полон запинок, пауз, фраз оборванных на середине или даже полностью отмененных последующими словами. С точки зрения компьютера такой разговор подобен смерти, для нас же это естественное положение вещей.

Суммируя, можно сказать, совершение ошибок есть естественное занятие человека. А раз ошибки естественны, значит система, неспособная сама их обнаружить и исправить, порочна. Таким образом, человеческих ошибок не бывает. Бывают ошибки в проектировании систем.

Сам термин «человеческая ошибка» до сих пор существует только по двум причинам. Во-первых, люди в ошибках системы склонны винить себя, поскольку по собственному эгоцентризму полагают, что подобные вещи происходят только с ними. Во-вторых, существующее положение вещей очень выгодно всякому руководству: гораздо легче уволить кого-либо, нежели признать, что система спроектирована плохо (и заплаченные за неё деньги «тю-тю»). Я, тем не менее, буду использовать этот термин и в дальнейшем. Но учтите, что под словосочетанием «человеческая ошибка» я буду иметь в виду только «действие пользователя, не совпадающее с целью действий этого пользователя»[1].

1. С ориентированной на счастье пользователей точки зрения, это единственная возможная формулировка.

Так что, когда в следующий раз прочтете в газете о том, как диспетчер посадил самолет вверх ногами, пожалейте не только пассажиров с экипажем. Пожалейте ещё и диспетчера. Вероятнее всего, ему ломают жизнь ни за что.

Типы ошибок

Таксономий[1] человеческих ошибок существует великое множество, чуть ли не каждый автор, пишущий на эту тему, создает новую таксономию. Не обошла эта привычка и меня. Итак, наибольшее количество человеческих ошибок при пользовании ПО раскладывается на четыре типа (сильно упрощенно, разумеется):

- **Ошибки, вызванные недостаточным знанием предметной области.** Теоретически, эти ошибки методологических проблем не вызывают, сравнительно легко исправляясь обучением пользователей. Практически же, роль этих ошибок чрезвычайно велика – никого не удивляет, когда оператора радарной установки перед началом работы оператором долго учат работать, и в то же время все ожидают должного уровня подготовки от пользователей ПО, которых никто никогда ничему целенаправленно не обучал. Еще хуже ситуация с сайтами, у которых даже справочной системы почти никогда не бывает (см. «Обучение работе с системой» на стр. 22).
- **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.** Фактически, количество этих ошибок пренебрежимо мало, к сожалению, недостаточно мало, чтобы вовсе их не засчитывать. 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 никогда не достигает нуля. Соответственно, единственным средством избежать этих ошибок является снижение требований к точности движений пользователя.

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

Тут уместно упомянуть одно из важных понятий инженерной психологии, а именно бдительность, т. е. способность оператора в течение продолжительного времени направлять существенную часть своего

1. Таксоно'мия [гр. taxis расположение по порядку + nomos закон] – теория классификации и систематизации сложноорганизованных областей действительности, имеющих обычно иерархическое строение.

внимания на состояние системы. Как показывает практика, ни один человек не способен долгое время обеспечивать бдительность без существенных потерь: мозг стремится найти себе более интересное занятие, отчего накапливается усталость и стресс. Нечего и говорить, что это никаким образом не приводит к получению удовольствия при работе с системой. Но как только бдительность снижается, количество ошибок возрастает в разы. Таким образом, проблема состоит в том, что для успешного пользования любой системой необходима определенная степень бдительности, но эта же бдительность пользователям неприятна. В условиях невозможности отбора пользователей (есть люди, способные быть бдительными дольше других), эта проблема не решается вообще. Потенциально, в систему можно ввести индикатор опасности текущего состояния, при этом пользователь имеет право быть не слишком бдительным большую часть времени, но зато получает и обязанность быть максимально собранным, когда горит «красная лампочка». С некоторыми оговорками, этот метод может быть использован (и используется) на атомной станции, но совершенно непонятно, как его можно применить, например, в электронной таблице, в которой всего два действия пользователя способны испортить целый документ. Важно также понимать, что крайне ценная возможность последующей отмены деструктивных действий (Undo) сама по себе не служит уменьшению количества человеческих ошибок, но помогает только уменьшить урон от них.

В действительности надо стремиться минимизировать количество ошибок, поскольку только это позволяет сберечь время (т. е. повысить производительность) и сделать пользователей более счастливыми за счет отсутствия дискомфорта.

Суммируя, при борьбе с ошибками нужно направлять усилия на:

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

Дополнительно к этим трём направлениям, есть и четвертое: снижение чувствительности системы к ошибкам. Для этого есть три основных способа, а именно:

- блокировка потенциально опасных действий пользователя до получения подтверждения правильности действия
- проверка системой всех действий пользователя перед их принятием
- самостоятельный выбор системой необходимых команд или параметров, при 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, о чём 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 получить можно. Для этого только надо требовать подтверждения не после команды пользователя, а до неё.

Предположим, чтобы удалить файл, нужно сначала в контекстном меню выбрать команду Разблокировать, после чего выбрать этот же файл и запустить процесс его удаления (неважно, с клавиатуры или из меню). В этом случае от пользователя действительно требуется подтвердить удаление, поскольку эти два действия напрямую не связаны друг с другом – если в одном из них была допущена ошибка, файл удалить не удастся.

К сожалению, этот принцип применять довольно тяжело. Дело в том, что ситуации, подобные описанной, встречаются довольно редко. Гораздо чаще приходится защищать не отдельные объекты (файлы, окна и т.п.), но отдельные фрагменты данных (например, текст и числа в полях ввода). Проблема состоит в том, что понятного и удобного элемента управления для этой цели нет. Единственным выходом служит скрытие потенциально опасных данных от пользователя до тех пор, пока он сам не скомандует системе их показать. Выход же этот отнюдь не идеальный, поскольку некоторым пользователям никогда не удастся понять, что, помимо видимых, 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 (см. «Структура окна» на стр. 93), 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[1].

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 (это вообще достоинство любого меню, см. стр. 76). Во-вторых, если действие запускается непосредственным манипулированием объектами, можно индицировать возможные действия изменением поведения этих объектов. Например, если бы форматирование диска запускалось не нажатием кнопки, а перенесением пиктограммы диска в область форматирования, можно было бы показывать пользователю, как с выбранного диска исчезают все файлы и папки. При этом не только снизилась бы вероятность ошибочного форматирования диска, поскольку перенести объект в другую область труднее, чем просто нажать на кнопку, но при этом исчезла бы необходимость предупреждать пользователя о грядущей потере данных с помощью дурацкого сообщения (подробнее об этом см. «Вон отсюда, идиот!» на стр. 40).

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

1. 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.

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[1].

В большинстве ОС есть специальный элемент управления, именуемый крутилкой (spinner, см. стр. 74). 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

Но что делать, если пользователь ввёл некорректное число с клавиатуры? Ответ прост. Надо индицировать возможную ошибку заменой цвета фона этого элемента на розовый.

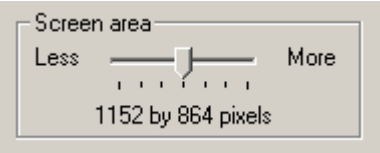


Рис. 3. Ползунок.

В тех же случаях, когда количество возможных значений невелико, лучше использовать другой элемент управления – ползунок (см. стр. 75). Мало того, что он позволяет устанавливать только определенные значения (с этим справился бы и раскрывающийся список или комплект переключателей), но он позволяет пользователю видеть взаимосвязь возможных значений и при этом использование этого элемента понятно даже новичку.

Самостоятельный выбор команд

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

Проиллюстрировать сферу применения данного метода удобно на примере печати. MS Windows User Experience заставляет использовать только один способ что-либо напечатать. Существует две команды меню Файл, а именно Печать и Параметры печати. Обе команды вызывают одноименные диалоговые окна. Проблема заключается в том, что обилие элементов управления замедляет[2] восприятие этих окон и увеличивает вероятность ошибки.

1. Никто не мешает также не позволять пользователю вводить текст в поля ввода, предназначенные для чисел и обратно.
2. Т. н. закон Хика: среднее время выбора равновероятных возможностей есть логарифм количества этих возможностей.

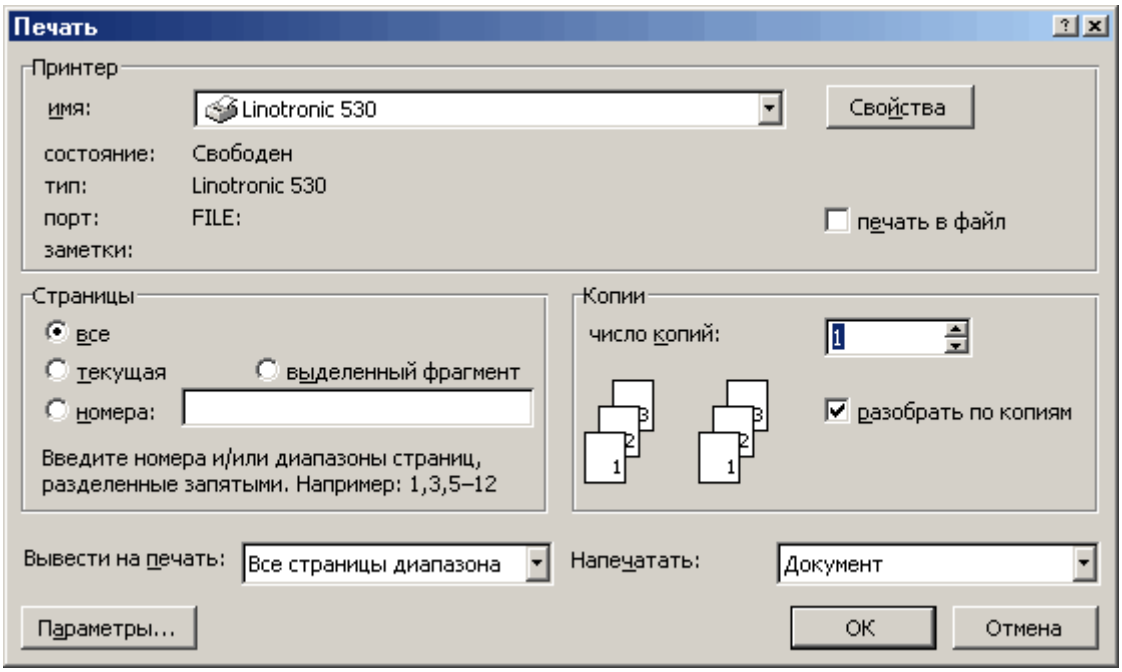


Рис. 4. Диалоговое окно печати в MS Word. © Microsoft.

Разберём это подробнее. Итак, чем меньше элементов управления, тем меньше вероятность ошибки. Система может уменьшить число элементов, если она знает сама, какими именно параметрами она должна руководствоваться.

Главной причиной появления этих диалоговых окон является печать нескольких копий. Причем есть простая зависимость – «количество копий обратно пропорционально частоте печати такого количества», т. е. сто копий печатают примерно в сто раз реже, чем печатают одну копию. Стандартное диалоговое окно печати содержит также область выбора принтера из числа установленных в системе. Большинство же пользователей имеет только один принтер. Так зачем заставлять это большинство каждый раз вдумчиво воспринимать совершенно не нужные им элементы интерфейса?

И так далее.

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

А теперь представьте себе другой вариант. В меню есть те же две команды – Печать и Параметры печати. Выбор первой команды вызывает немедленную печать документа с текущими настройками. Выбор второй вызывает диалог со всеми доступными параметрами печати, т. е. в системе с одним принтером никогда не появится ненужная группа элементов.

Если же пользователь начинает проявлять буйную активность и печатать несколько копий разом, включается другой механизм. В первый раз, когда пользователь меняет число копий в окне настроек печати, программа запоминает его действие и при следующем выборе команды Печать выводит диалоговое окно со всего двумя элементами управления – полем ввода, в котором уже стоит число копий (которое было запомнено в предыдущий

раз) и кнопкой OK. Поскольку программа не может быть уверена в правильности числа копий, цифру лучше всего выводить нестандартным цветом, чтобы привлечь внимание пользователя.

И так до тех пор, пока пользователь два раза подряд не введет единицу в поле ввода (что переводит его в разряд представителей большинства) или не введет новое число копий (каковое и будет запомнено). Причём такая метода применяется абсолютно ко всем возможным настройкам, а не только к числу копий. Таким образом, большинство пользователей становится счастливо, а количество ошибок сокращается, что хорошо.

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

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

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

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

Два уровня ошибок и обратная связь

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

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

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

С ошибками из четвертого пункта всё ясно. Всякий раз, когда мы теряем возможность, по крайней мере, проверить валидность данных или самой системы, мы вступаем на слишком уж скользкий путь. Межпланетные зонды, из-за ошибок в ПО улетающие не туда, коммерческие договоры, в которых обнаруживаются ошибки, ошибочные номера телефонов в записной книжке – всё это примеры неисправляемых ошибок. Разумеется, такие ошибки всегда обнаруживаются, проблема в том, что к моменту их

обнаружения становится поздно их исправлять. Именно поэтому такие ошибки гораздо хуже ошибок, которые исправить трудно, но которые, по крайней мере, сразу видны. Единственной индустрией, научившейся извлекать пользу от необнаруженных ошибок, является производство почтовых марок – марки с опечатками стоят у филателистов многократно дороже марок без оных.

Это было знание программистов. Теперь пора перейти к интерфейсу и определить, почему ошибки первого типа («исправляемые во время») гораздо лучше ошибок второго типа («исправляемых после»).

Вообще говоря, объяснение этого факта двояко. Объяснение есть как субъективное, так и объективное, 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 из предыдущей главы, любое действие пользователя состоит из семи шагов (см. «Длительность интеллектуальной работы» на стр. 5). Всякий раз, когда пользователь обнаруживает, что он совершает ошибку, ему приходится возвращаться назад на несколько этапов. Более того, чтобы исправить совершенную ошибку, от пользователя требуется:

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

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

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

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

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

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

Обучение работе с системой

В традиционной науке о человеко-машинном взаимодействии роль обучения операторов чрезвычайно велика. Мало того, что в дополнение к самой системе разрабатывается методология обучения её будущих пользователей, так еще и создаются 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. 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. Популярно говоря, пользователя можно научить становиться на задние лапки за кусочек сахара (который есть стимул), но научить его перетаскивать тяжеленные камни за тот же кусочек сахара нельзя, потому что награда не стоит усилий.

Применительно к компьютерным системам этот закон действует без каких-либо исключений. Обучение есть действие: если обучаться легко, пользователям будет достаточно слабого стимула, если тяжело, стимул придется увеличивать. Но если стимул для пилота самолета получают преимущественно командными методами (если он не научится определенному минимуму, его просто не допустят до работы), то в случаях компьютерных систем стимул есть вещь почти исключительно добровольная. Это значит, что пользователь обучится пользоваться программой или сайтом только в том случае, если он будет уверен, что это 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, причем не только безучастно сообщать о наличии той или иной функции, но и расхваливать кусочки сахара, которые пользователь получит, этой функции обучившись. 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

Обычно считается, что в случае ПО есть два способа повысить эффективность обучения (помимо метода «обучения плаванию посредством выбрасывания из лодки»), а именно бумажная документация и «оперативная справка». Это категорически неправильно. Во-первых, способов много, и самые главные способы в эту систему не попали. Во-вторых, одно из понятий, входящих в эту таксономию, просто некорректно: оперативность справки зачастую бывает отрицательной (т. е. поиск в ней занимает больше времени, чем поиск того же в бумажной документации). Поэтому разумно привести более совершенный список средств обучения:

- общая «понятность» системы
- обучающие материалы.

А теперь можно разобрать эти составляющие по отдельности.

Понятность системы

Термин «понятность», будучи крайне приятным и во многих случаях удобным, на самом деле нехорош, поскольку очень уж размыт[1]. Однако кое-что выделить можно, а именно ментальную модель, метафору, аффорданс и стандарт (да и то окажется, что они сильно смыкаются). Обратите внимание, что хотя в этой главе говорится о понятности, изложенные свойства, будучи хорошо реализованы, также существенно снижают количество человеческих ошибок.

Ментальная модель

Зачастую, или, точнее, почти всегда, чтобы успешно пользоваться какой-либо системой, человеку необходимо однозначно понимать, как система работает. При этом необязательно точно понимать сущность происходящих в системе процессов, более того, необязательно понимать их правильно. 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. Так, например, человек, привезенный на машине времени из прошлого и никогда не встречавшийся с электричеством, поставит утюг на плиту[2], отчего прибор непременно сгорит.

Другой пример, на этот раз из предметной области: файловая система. Каждый, кто обучал кого-нибудь пользоваться компьютером, знает, как трудно объяснить пользу от записи файла после его редактирования. Дело в том, что без понимания сущности происходящих в компьютере процессов понять сущность записи невозможно. Допустим, пользователь создал новый документ, что-то в нем сделал, после чего попытался выйти из программы. Программа спрашивает его «Сохранить документ или нет?»; тут и начинается самое интересное.

Во-первых, в этом вопросе главное значимое слово непонятно. Что такое сохранить? Где сохранить? Куда сохранить? Если же вопрос ставится техническим языком, а именно «Записать документ на диск?», то и здесь получается гадость: на какой такой диск?

Во-вторых, что важнее, даже если пользователь понял вопрос, он все равно не может понять, зачем документ сохранять? Как-никак документ уже имеется, сохранять его не надо.

Проблема усугубляется тем, что даже самый начинающий пользователь знает, что если он нажмёт кнопку Ок, начнется какое-то действие. А поскольку пользователь не хочет, чтобы действие, которого он не понимает, начиналось, он недрогнувшей рукою нажимает кнопку Нет, после чего программа закрывается, а файл не сохраняется.

Иной аспект той же проблемы: как научить пользователя превентивно сохранять рабочий файл время от времени, чтобы, когда компьютер зависнет, можно было встретить судьбу во всеоружии? Сделать это практически невозможно, поскольку пользователь искренне уверен, что если файл есть в компьютере, с ним уже ничего не сделается.

В результате, есть только два способа научить пользователя сохранять файлы. Сущность первого способа состоит в научении пользователя, что если он не будет время от времени и перед выходом из программы сохранять файл, у него будут проблемы. Рано или поздно пользователь начнет

1. Особо неприятен в этом смысле термин «интуитивная понятность», не имеющий, вообще говоря, никакого смысла. Чуть ли не единственными артефактами, понятными *интуитивно*, являются соска и лестница. Всему остальному надо учиться.
2. Особняком стоит т. н. «синдром электрочайника», который объясняется сугубым автоматизмом действия.

сохранять файл автоматически, проблема в том, что произойдет это скорее поздно, чем рано, и вполне может оказаться, что и никогда. Этот способ обладает определенным своеобразием: пользователь не поймет, *зачем* он это делает, он будет только знать, что делать это *надо*. На жаргоне этнографов и психологов такое поведение называется ритуалом. Получившийся в результате такого обучения ритуал не будет принципиально отличаться от, скажем, плясок каннибалов вокруг волшебного дерева Тум-Тум, чтобы не было эпидемии: пользователь будет нажимать кнопку Сохранить в качестве жертвы Великому Энжэмэдэ, Богу Компьютера. К слову сказать, пользователей, которые пользуются компьютером с помощью ритуалов, не так уж мало.

Второй способ: пользователю можно объяснить, что в компьютере есть две подсистемы памяти, одна постоянная, а другая временная; при выключении или при зависании компьютера содержимое временной памяти теряется, так что если документ нужен будет и позже, его надо перенести в постоянную память; перенос туда документа называется сохранением. Это тоже непросто, поскольку пользователь уверится, что люди, придумавшие компьютеры, круглые дураки (ему неинтересны технологические причины), но зато это знание научит его сохранять файлы. Разумеется, он ещё не раз забудет сохранить файл, но это произойдет из-за недостатка внимания.

Так вот, изначально пользователь не имеет ментальной модели памяти компьютера. Проблема же состоит в том, что компьютер не дает ему возможности самому построить модель, так что единственным её источником может являться обучение. Это один из самых больших недостатков дизайна современных компьютеров, во всяком случае, первый компьютер без разделения памяти на постоянную и временную (Palm Pilot) разошелся тиражом в миллионы экземпляров.

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. Но творческий подход может помочь.

Избегайте создавать элементы управления, функциональность которых зависит от контекста

Существует, однако, одно простое правило: поскольку элементы, выполняющие несколько разных функций в зависимости от контекста, существенно усложняют построение ментальной модели, их лучше не создавать. Поэтому слишком много элементов управления обычно делать лучше, нежели слишком мало элементов, во всяком случае, опасно ставить перед собой цель «во что бы то ни стало сократить количество элементов».

Метафора

Как было сказано, чтобы научиться пользоваться системой, пользователю нужно построить ментальную модель этой системы. Очень хочется избавить его и от этой работы. Лучшим способом добиться этого является применение 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)[1].

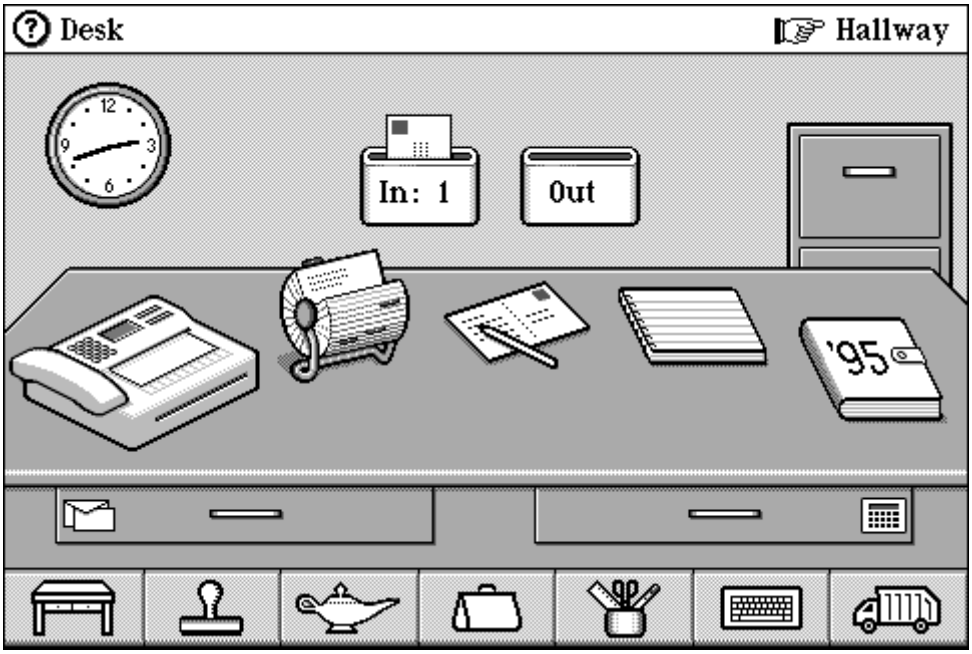


Рис. 5. Главный экран ОС Magic Cap. Будучи вся построена на метафорах, ОС приобрела широчайшую известность среди журналистов компьютерных изданий (они сразу всё понимали). С другой стороны, она не смогла добиться такой же любви у пользователей: они её понимали, но отказывались с ней работать из-за её неэффективности. Классический пример горя от ума. © General Magic © Susan Kare

К сожалению, метафоры отнюдь не безоблачны. У них есть несколько существенных недостатков.

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. Однако через некоторое время

1. Данный пример интересен ещё и тем, что в игру вступает стандарт (о котором позже). Поскольку все разработчики программ для проигрывания звуков копировали в интерфейсе материальные проигрыватели, образовался стандарт на программы такого рода. Теперь пользователям не обязательно обучаться значению кнопок именно по физически существующим проигрывателям, для этого подходит и любая программа.

появился не копирующий гранки QuarkXpress, и отвоевал большую часть пользователей лишь потому, что работал более эффективно, не таская на себе груз старой идеи. Пользователи предпочитали потратить больше времени на обучение, зато выиграть в скорости работы. Анекдотичность ситуации заключается в том, что к настоящему времени гранки не используются вовсе, появилось поколение пользователей, которое никогда их в глаза не видело. Результат: обучаясь PageMaker, они должны дополнительно обучаться идее гранок, которая им вовсе не нужна.

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. Не надо забывать, что большинство систем, сильно базирующихся на метафоре, проиграли конкурентам. Таков уже упоминавшийся PageMaker, таковой была операционная система Magic Cap, таковой была оболочка MS Bob, в которую было вложено множество денег, и которая была прикрыта после нескольких месяцев микроскопических продаж (это самые шумные падения, были и другие).

Аффорданс

Другой составляющей понятности является нечто, по-английски называемое affordance[1]. Мне не удалось удовлетворительно перевести[2] этот термин (а до меня никто и не пытался), так что это понятие в книге будет называться неоригинально: «аффорданс».

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

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

1. Термин введен Дональдом Норманом (Donald Norman. The Design of Everyday Things).
2. Частично в качестве перевода подходит слово *наглядность*, но из-за того, что в нём упор ставится на зрение (глядеть), предпочесть его я не решился.

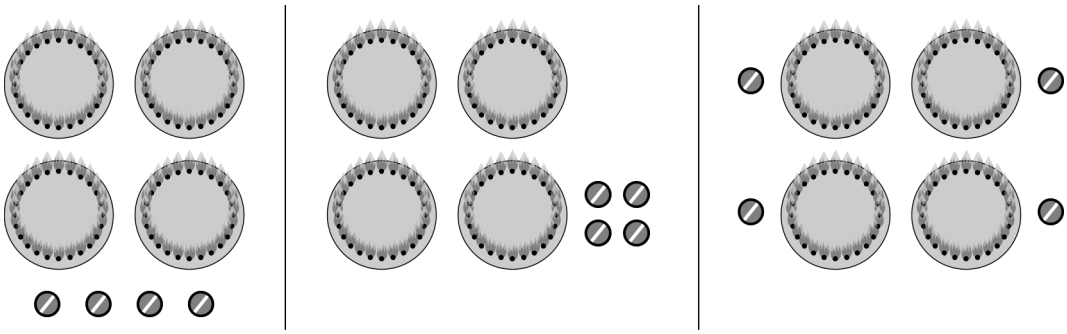


Рис. 6. Отсутствие аффорданса в дизайне кухонной плиты. Слева стандартный вариант, недостаток которого заключается в том, что невозможно умозрительно определить, какой диск управляет какой конфоркой. В центре и справа варианты с аффордансом, не имеющие этой проблемы, при этом работающие по-разному. В 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. Область эта еще практически не разработана, так что она сулит, при удаче, большой успех и славу. Надеюсь, что вы их достигнете.

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[1].

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

Таким образом, чтобы стандарт заработал, он должен быть популярен. 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. С другой стороны, этот стандарт оставляет неопределенным очень многое (никто да не обнимет необъятного), и это многое в разных системах трактуется по-разному. Бесполезно пытаться найти общий знаменатель во всех системах, гораздо эффективнее установить собственный стандарт, не противоречащий стандарту ОС, но дополняющий его; после чего этого стандарта надо придерживаться.

Последовательность в реализации интерфейса есть первое условие качества результата

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

1. Один из моих читателей рассказал мне по этому поводу поучительную историю. Во время разговора с девушкой о космосе, читатель показал девушке Вегу, большую, голубую звезду. При этом девушке было сказано, что если на место Солнца поместить Вегу, то орбита Земли будет находится все еще внутри самой звезды. Услышав это сообщение, девушка приобрела уверенность, что все большие звезды есть звезды холодные, мотивировав это тем, что, дескать, кран с холодной водой обозначается синим.

Обучающие материалы

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

Что нам нужно и что у нас есть

Количество подсистем справки, нужных для того, чтобы пользователь научился пользоваться системой, довольно невелико, так что все их можно легко разобрать[1]. Когда я говорю «подсистема справки», я имею в виду часть справочной системы (или документации), которая выполняет сугубо определенные функции и требует сугубо определенных методов представления.

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

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

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

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

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.

1. Обратите внимание, что это моя собственная классификация, вызванная к жизни тем, что большинство существующих классификаций справочных систем слишком тяжеловесны и размазаны.

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

Система должна индицировать все свои состояния

Сообщения об ошибках. Это тема настолько многогранная, что о ней отдельно (см. «Каким должно быть сообщение об ошибке» на стр. 42).

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

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

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

Структурированная электронная документация. Плохо предназначена для чтения больших объемов материала, зато обеспечивает легкий поиск и не имеет лимита объема. Занимает большой объем пространства экрана. Плохо подходит для показа крупных изображений, зато в неё могут быть легко интегрированы видео и звук.

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

Рис. 7. Пример пространства интерфейса, выделенный на показ справочной информации. Обратите внимание, что даже обычная подпись к полю ввода также является таким пространством.

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

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

Базовая и обзорная справки

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

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

Справка предметной области

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

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

Процедурная справка

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

Контекстная справка

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

Спиральность

В отличие от художественной литературы, справочные системы не предназначены для того, чтобы приносить 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. Исследования[1] 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 за последние сто лет не выросли. Логичные в таких условиях интерфейсы в эстетике художника Шишкина по меньшей степени противоестественны. С другой стороны,

1. Например, E. Hollnagel, «Keep Cool: The Value of Affective Computer Interfaces in a Rational World,» Proc. HCI Int'l 99, vol. 2, Lawrence Erlbaum Associates, Mahwah, N.J., 1999, pp. 676–680 и S.W. Draper, «Analyzing Fun as a Candidate Software Requirement», Personal Technology, vol. 3, no. 1, 1999, pp. 1–6.

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).

Рис. 8. Классический стул Vitra. модель .03. Для стульев очень важно не только удобство сидения и эстетическая привлекательность, но и удобство хранения. Поскольку складные стулья обычно выглядят неважно, дизайнерам приходится изощряться, придумывая стулья, способные собираться в «стопки». Дизайнерам интерфейсов также частенько приходится добиваться свойств продукта не только сугубо утилитарных, но и к тому же сравнительно неочевидных. © Дизайнер Маартен Ван Северен. © Vitra.

- 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, коммуникационный и промышленный дизайны. Итак, какие их принципы могут быть использованы в дизайне интерфейса?

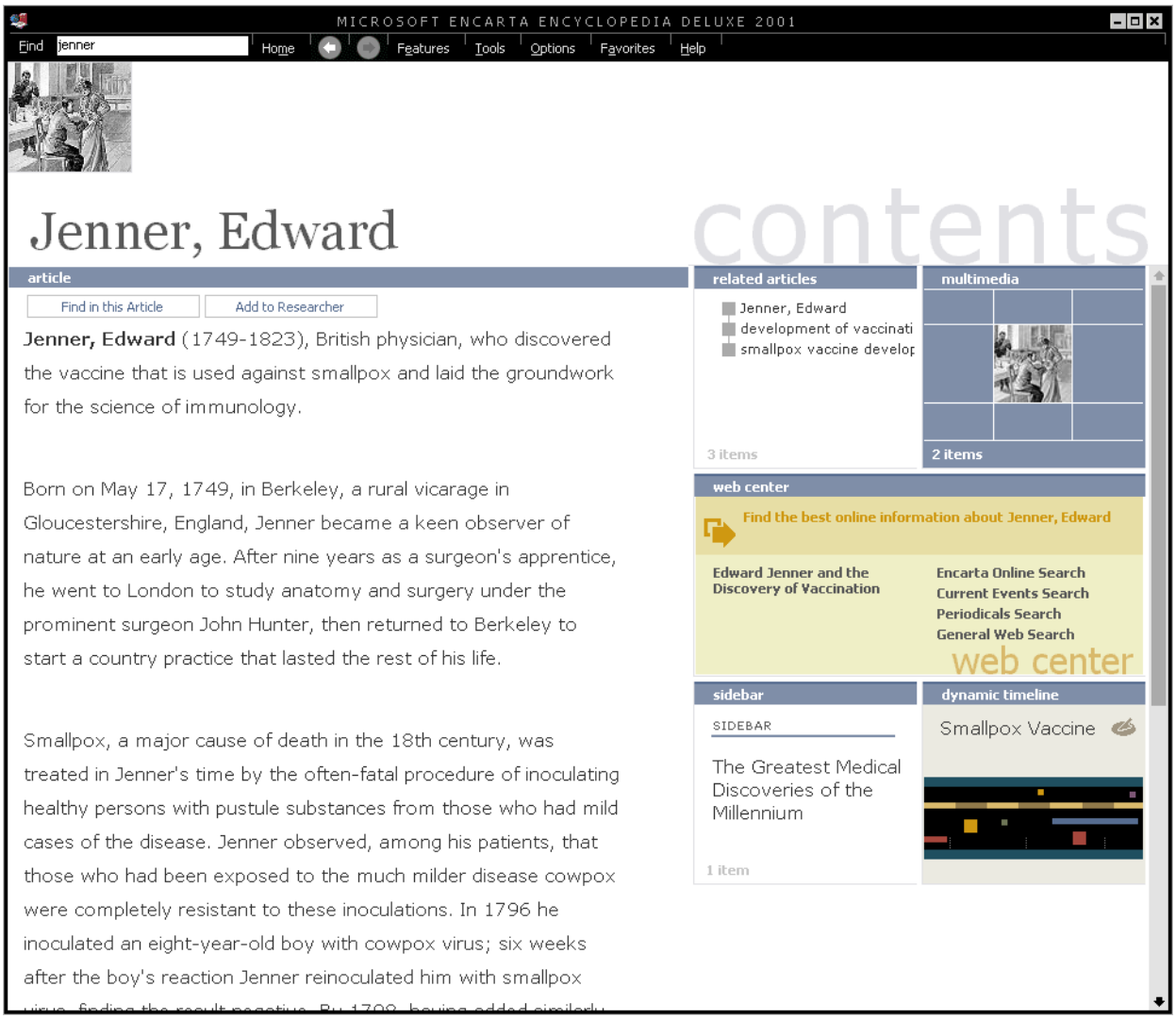


Рис. 9. Интерфейс MS Encarta Encyclopedia 2001. Замечательный пример дизайна – интерфейс равно удобный и эстетически привлекательный. История этого интерфейса была весьма поучительна – с каждой новой версией интерфейс становился все скромнее и скромнее (энциклопедия выходит каждый год, пик красоты пришелся на первую версию). Обратите внимание на обилие пустого пространства и ограниченное использование цвета. Несмотря на внешнюю пустоту окна, все функции доступны одним нажатием кнопки. © Microsoft.

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.

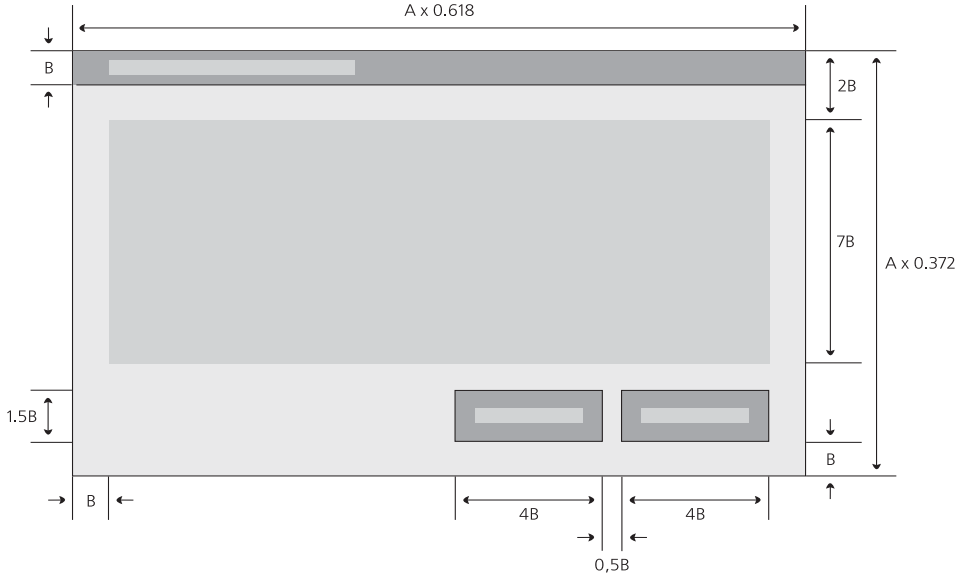


Рис. 10. Пример закономерностей в диалоговом окне. Старайтесь минимизировать количество констант (тем более, что двух констант обычно хватает на все). Разумеется, единожды примененных закономерностей необходимо придерживаться во всей системе.

Красота понятие относительное[1]. 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 осознаются 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

1. Иначе – классовое.

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 полю 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 существенный.

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, обнаружил, что *весь* мой опыт полезен 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 начале 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 (см. «Кратковременная память» на стр. 47)ы.
- Разделение крупных действий пользователей на более мелкие. При этом количество работы увеличивается, но зато субъективная длительность снижается. Плох этот метод тем, что увеличивает усталость.

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

По острию ножа

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

Дело в том, что почти всё время пользователь может что-либо испортить и знает это. Он может отформатировать жесткий диск, может стереть или испортить нужный файл. Неудивительно, что пользователь часто боится. А если не боится, то склонен недооценивать свои возможности к разрушению, отчего снижается бдительность. Куда ни кинь, всюду клин.

Единственным полноценным решением является возможность отмены пользователем своих предыдущих действий, без ограничения количества уровней отмены и типа отменяемых действий[1]. 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 что ожидать скорого наступления эры всеобщего счастья не приходится.

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

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

Таким образом, пользователей нужно всемерно снабжать ощущением, что ничего не может произойти, пока этого не захочется самому пользователю. Функции, работающие в автоматическом режиме, но время от времени просыпающиеся и требующие от пользователей реакции, вызывают стресс. В любом случае, стоит всеми силами внушать пользователям мысль, что только явно выраженное действие приводит к ответному действию системы (это, в частности, главный аргумент против ролловеров[2] – пользователь ещё ничего не нажал, а уже что-то произошло).

1. Сейчас нормой является невозможность отменить свои действия после записи файла.
2. Термин трудной судьбы. Также «ховер». Обозначает объект, изменяющий свой вид при наведении на него курсора, показывая, что с объектом можно совершить какое-либо действие.

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

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

- недостаточно гибка, чтобы приспособиться к его действиям
- недостаточно умна, чтобы показать ему его возможные границы его действия
- самоуверенна и считает, что пользователь дурак, которым можно и нужно помыкать.

Разберем это подробнее.

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

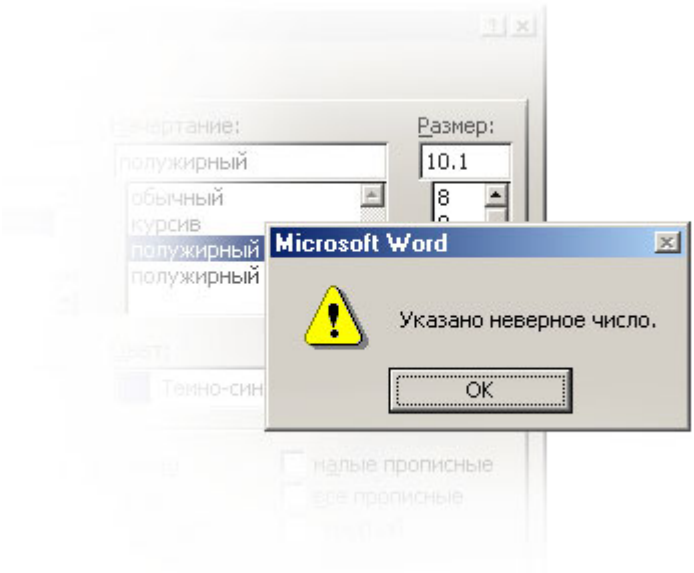


Рис. 11. Вот что бывает, если пользователь попытается ввести значение, которое ему нужно, но которое система не умеет обрабатывать. Тут возможно три альтернативных решения. Во-первых, при проектировании системы можно более тщательно подходить к выбору её функциональности. Во-вторых, можно просто игнорировать неправильность значения, округляя его до ближайшего возможного (индицируя, возможно, самостоятельность действий системы однократным миганием соответствующего поля ввода). В-третьих, вместо обычного поля ввода можно использовать крутилку (см. стр. 74). © 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.

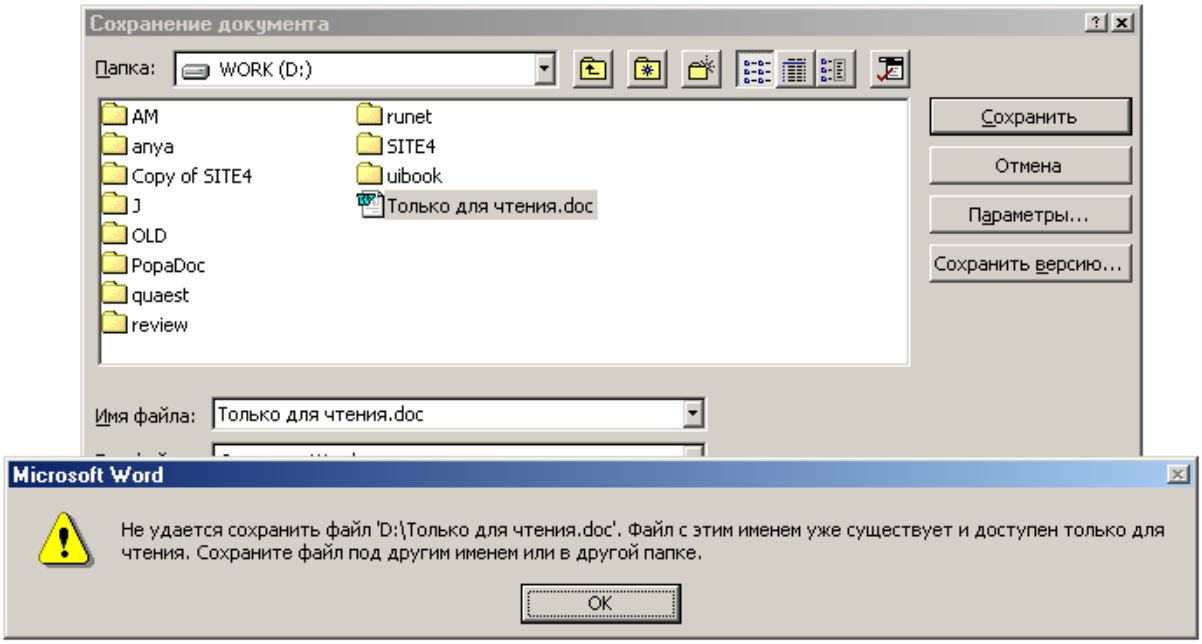


Рис. 12. Типичное сообщение об ошибке, вызванное нежеланием системы показать пользователю границы его действий. С одной стороны, оно разумно – файл не может быть записан под этим именем. С другой стороны, это сообщение показывается пользователю каждый раз при попытке перезаписать такой файл. Если бы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

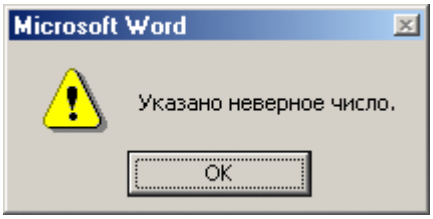


Рис. 13. Для кого неверное? И кто, собственно, виноват, система или пользователь? © Microsoft.

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, их ненавидят.

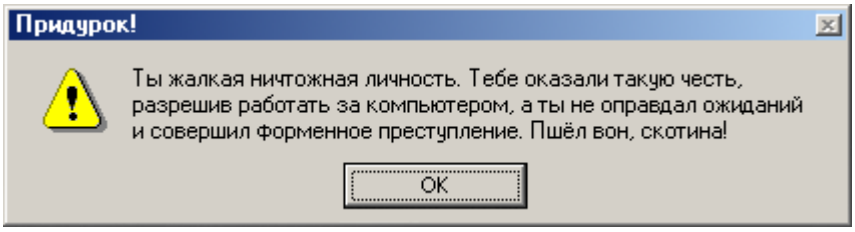


Рис. 14. Именно так пользователи воспринимают любые сообщения об ошибках.

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 редки, так что их чаще всего 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:

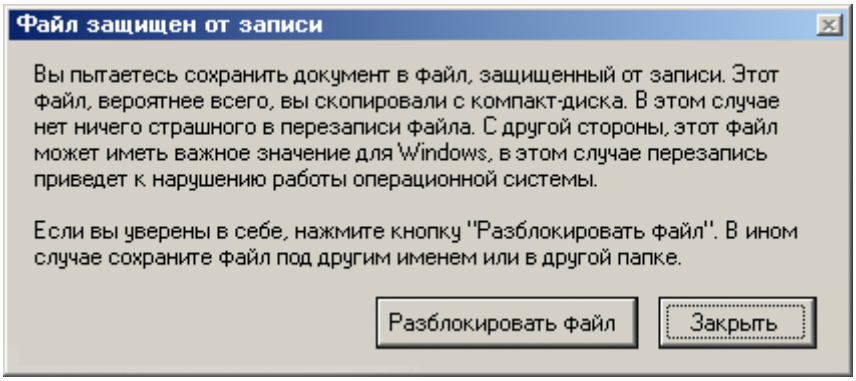


Рис. 15. Улучшенное сообщение об ошибке. Обратите внимание, что кнопка Закрыть выбрана по умолчанию, чтобы снизить вероятность перезаписи важных файлов. Конечно, лучше всего было бы, чтобы ОС сама снимала с копируемых с компакт-диска файлов метку Read Only. Многие проблемы при этом бы исчезли, поскольку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

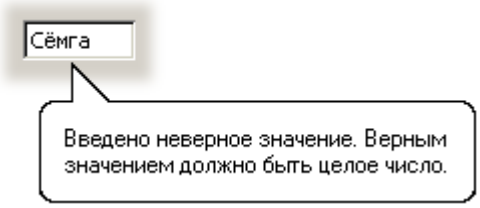


Рис. 16. Пузырь в его лучшем проявлении (не обращайте внимания на текст).

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

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

Пароли

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

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

Для этого есть несколько путей. Во-первых, от них можно отказаться вообще. Этот путь почти всегда предпочтителен, проблема в том, что он вызывает существенное отторжение у сетевых администраторов (которые все как один параноики). В самом деле, зачем требовать от пользователя пароль, если подобрать этот пароль злоумышленник сможет без труда? И обратно: поскольку единственным способом создания действительно трудноподбираемых паролей является генератор случайных чисел, каковые числа (символы, по необходимости) невозможно запомнить, пользователи либо будут записывать их на бумажке (в этом случае пароли еще легче украсть), либо будут забывать, что дорого стоит. Я, например, вынужден пользоваться четырьмя действительно случайными паролями, причем тремя из них не реже раза в неделю. При этом, несмотря на то, что я пользуюсь этими паролями не менее года, я не помню *ни одного*. В результате, все четыре пароля записаны у меня на бумажке, которую увидеть может любой (я её не прячу). Какая уж тут защита?

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

Этот метод хорош, когда дело касается защиты информации. Однако гораздо чаще, особенно в интернете, важна не столько защита информации, сколько идентификация пользователя (которая может сопрягаться с защитой, а может и не сопрягаться). В этом случае от паролей отказаться невозможно, просто потому, что не существует другой, столь же эффективной, технологии идентификации. При этом содержимое пароля как таковое не очень важно, важно его отличие от паролей других пользователей

(обратите внимание, что в этом случае само по себе *имя пользователя* является паролем). Обычно в таких условиях при регистрации применяют стандартную группу элементов «имя, пароль, подтверждение пароля», после чего, уже при нормальной деятельности, пользователь получает возможность ввести пароль только в первый раз (см. рис. 17).

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

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

Таким образом, эффективнее всего максимально здраво определить степень важности защищаемой информации, после чего выбрать адекватную схему защиты, стараясь, чтобы она была максимально лёгкой для пользователей. Как правило, их субъективное удовлетворение важнее потенциального вреда от потери информации.

Самовыражение

Страсть к самовыражению является одной из самых сильных черт человеческой натуры. В большинстве случаев люди самовыражаются через вещи (двигают мебель и покупают модную одежду), когда нет вещей – насвистывают или поют изящные песни «лишенным приятности голосом». Ни один человек не может существовать сколько-нибудь продолжительное время в обстановке, не допускающей самовыражения.

Среднестатистический человек проводит в день около восемнадцати минут в туалете, этого времени достаточно, чтобы вызвать острое желание выразить себя: покупается особая, не такая как у других, вешалка для рулона туалетной бумаги, стены красятся в сравнительно уникальные цвета, а наиболее творческие «пользователи туалета» вешают на дверь зеркало и т.д. Неудивительно, что пользователи хотят выразить себя и в программах, которыми они пользуются. Соответственно, возможность настроить систему под свои нужды является мощной причиной субъективного удовлетворения.

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

С другой стороны, настроенный под свои нужды интерфейс, судя по всему, снижает усталость работников и повышает их рабочее настроение. В таких условиях потеря 45 минут в неделю может оказаться скомпенсированной благодаря повышению производительности труда.

Таким образом, в продуктах, продаваемых пользователям напрямую, возможность настройки под конкретного пользователя обязательна. Во всех остальных случаях, судя по всему, нужен способ настройки, позволяющий максимально изменить вид системы минимумом команд (чтобы снизить время, затрачиваемое на самовыражение). Хорошим вариантом является банальный выбор варианта готовой настройки из списка без возможности модифицировать встроенные варианты.

Всячина

В этой главе описываются вещи, которые, не относясь напрямую к основным характеристикам интерфейса, либо очень важны для интерфейса, либо (по моим наблюдениям) слишком часто делаются неправильно.

Память

Возможности человеческой памяти существенно влияют на качество взаимодействия пользователя с системой. Для нас актуально знать про две подсистемы памяти, а именно про кратковременную и долговременную подсистемы.

Кратковременная память

Свойства, а точнее ограничения кратковременной памяти (КВП) являются очень важными факторами при разработке интерфейса. Дело в том, что вся обработка поступающей информации производится в КВП, в этом кратковременная память сходна с ОЗУ в компьютерах. Сходство, однако, не является полным, так что думать о КВП, как об ОЗУ, я не советую.

Что попадает в КВП. Удобно, хотя и неправильно, считать, что в КВП попадает всё, что кажется нужным и имеющим какой либо смысл. Соответственно, чтобы что-либо попало в КВП пользователя, пользователь должен это заметить (для чего, собственно говоря, и полезно проектировать интерфейс с учетом возможностей человеческого восприятия) и счесть полезным лично для себя. Как правило, для опытного пользователя оценка полезности не представляет проблем, неопытные же почти всегда суют в КВП наиболее заметные детали. Таким образом, самое важное в интерфейсе должно быть наиболее заметным (вот мы и узнали теоретическое обоснование очередного очевидного факта).

Изменение содержимого. Другая интересная особенность КВП заключается в том, что смена содержимого в ней происходит скорее из-за появления новых стимулов, нежели чем просто от времени. С одной стороны, это значит, что без новых стимулов КВП остается неизменной. С другой стороны, поскольку отсутствие стимулов есть труднодостижимый идеал, содержимое КВП постоянно меняется. Практический смысл этого наблюдения состоит в том, что нельзя допускать, чтобы пользователь отвлекался, поскольку новые стимулы при отвлечении стирают содержимое КВП. Но и это есть только мечта. Приходится удовлетворяться тем, чтобы максимально облегчать пользователю возвращение к работе (об этом более подробно см. «Потеря фокуса внимания» на стр. 8).

Объем КВП. Чуть ли не единственным правилом интерфейсной науки, известным широкой публике, является моветон, гласящий, что в группе чего угодно не должно быть более семи плюс-минус два элементов.

Проблема в том, что это правило имеет слабое отношение к реальности и его практическое значение невелико. Более того, оно даже вредно, поскольку его знание народом, усугубленное незнанием остальных правил, приводит к искренней уверенности, что если в интерфейсе не более девяти элементов (семь плюс два; люди предпочитают складывать, нежели вычитать), то этот интерфейс автоматически хорош. Что, мягко говоря, не совсем правильно. Но начнем сначала.

Оценивать объем КВП применительно к интерфейсу как всеобъемлющие 7±2 элементов не вполне правомерно. Во-первых, как уже было сказано, в КВП информация хранится преимущественно в звуковой форме.

Это значит, что вместо смысла запоминаемых элементов в КВП хранится текст, написанный на этих элементах. Для нас это означает, что подвергать ограничению следует преимущественно те элементы, которые содержат текст. Во-вторых, известно, что в память помещается гораздо больше, но только в тех случаях, когда элементы сгруппированы[1]. Соответственно, всегда можно сгруппировать элементы и поместить в КВП пользователя больше информации. В-третьих, существует некоторое количество людей, способных удержать девять значений в КВП, но количество людей, способных удержать в памяти только пять или шесть значений, тоже довольно существенно. Это значит, что с практической точки зрения гораздо удобнее считать, что объем КВП равен ровно семи элементам (или, если ситуация позволяет, шести), поскольку рассчитывать нужно не на сильное, а на слабое звено. И, наконец, самое важное. Информация не только хранится в КВП, она в нем же и обрабатывается. Это значит, что один этап обработки занимает место как минимум одного элемента КВП. Более того: контекст предыдущих действий тоже хранится в КВП, снижая доступный объем.

С практической точки зрения важно еще и следующее. Если на интерфейс смотрит опытный пользователь, почти вся необходимая ему информация содержится не в кратковременной, но в долговременной памяти, а значит, специально про КВП думать не надо. Более того, зачастую и для неопытных пользователей объем КВП не важен. Предположим, такой пользователь смотрит на раскрытое меню. Поскольку контекст его предыдущих действий никто не отменял, пользователь знает, чего он хочет, но не знает еще, как этого добиться. Он сканирует меню и, найдя наиболее многообещающий элемент, выбирает его, при этом ни один из ненужных ему элементов в КВП не попадает. Проблемы с КВП начинаются только тогда, когда ни один элемент не кажется ему более желанным, чем другие. При этом пользователю необходимо поместить все элементы меню в КВП и сделать выбор. Соответственно, большинство таких проблем может быть пресечено визуальной организацией элементов (чтобы убедиться, что пользователь смотрит именно в то меню, что ему нужно) и правильным наименованием отдельных элементов (чтобы по тексту элемента сразу была понятна его применимость). Впрочем, слишком большой объем элементов неопытным пользователям всё-таки вредит: если пользователь долго сканировал список в поиске нужного элемента, некоторая часть списка всё-таки попадет в КВП и испортит контекст, так что увлекаться большим набором возможностей тоже не стоит.

Так что значительно эффективнее считать, что объем кратковременной памяти равен пяти (шести, из которых один в запасе) элементам. Не более, но и не менее.

Нагрузка на КВП. В целом, использовать КВП пользователям неприятно. В этом заключается самая большая проблема КВП применительно к интерфейсу, большая даже, чем человеческие ошибки, вызванные выпадением элементов из памяти. Объясняется это неприятие КВП просто – и запоминание, и извлечение информации из памяти требует усилий. Более того. Поскольку содержимое КВП теряется при поступлении новых стимулов, пользователям приходится прилагать усилия, чтобы просто удержать информацию в памяти (вспомните, сколько раз вы повторяли номер телефона, чтобы удержать его в памяти на время, пока вы переходите в другую комнату).

Таким образом, необходимо снижать нагрузку на память пользователей, т. е. избегать ситуаций, когда пользователю приходится получать информацию в одном месте, а использовать её в другом. Лучшим способом достиже-

1. Человек не способен удержать в КВП 12 произвольных букв, но зато способен без ощутимого труда удержать 7 слов по 12 букв в каждом (т. е. 84 буквы). Попробуйте удержать в памяти комбинацию «чдепсеритаеи» и сравните свои усилия с удержанием слова «деепричастие». Такая смысловая группировка, при которой набор элементов превращается в единый элемент, позволяет на порядок увеличить объем КВП.

ния этой цели является непосредственное манипулирование (см. «Непосредственное манипулирование» на стр. 5), у которого, кстати, есть ещё множество других достоинств.

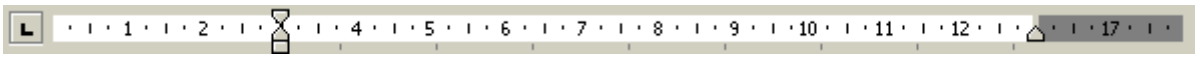


Рис. 18. Пример ослабления загрузки КВП непосредственным манипулированием (см. стр. 5) в MS Word. Перемещая ползунки, пользователь перемещает абзацные отступы у выделенного фрагмента текста. Без ползунков ему бы пришлось сначала помещать в КВП величину потребного смещения и алгоритм своих действий, затем, по мере сил, исполнять алгоритм и вводить значения в открывшееся диалоговое окно. Линейка позволяет избежать этого, давая к тому же хорошую обратную связь (см. «Два уровня ошибок и обратная связь» на стр. 20).

Вообще говоря, любой ввод параметров не значениями, а воздействием на управляющие элементы (т. е. верньеры) сильно снижает нагрузку на память. Другой разговор, что верньеры занимают много места на экране, плохо подходят для точного ввода значений и всегда оказываются хуже непосредственного манипулирования, поскольку при непосредственном манипулировании пользователям не нужно помещать в КВП даже и алгоритм действия.

Долговременная память

Долговременная память, вообще говоря, лучше поддается интроспекции, так что рассказ о ней будет короче. Как все мы знаем (и пока еще не забыли), объем ДВП очень велик, информация, попавшая в ДВП, хранится, судя по всему, вечно и так далее. С точки зрения дизайна всё это не представляет особого интереса. Интерес представляют два вопроса:

- При каких условиях информация попадает в ДВП?
- Сколько «стоит» вспоминание?

Оба вопроса очень интересны с точки зрения обучения пользователей (см. стр. 22), второй вопрос, к тому же, интересен еще и с точки зрения улучшения способности пользователей сохранять навыки работы с системой в течение длительного времени (а это одна из основных характеристик хорошего интерфейса).

Внутрь ДВП. Сейчас считается (и это мнение вряд ли будет изменено в дальнейшем), что информация попадает в ДВП в трех случаях. Во-первых, при повторении, т. е. при зубрежке. Во-вторых, при глубокой семантической обработке. В-третьих, при наличии сильного эмоционального шока («навсегда запомнила графиня последний взгляд полковника Поркбриджа. «Пуркуа па?», казалось, спрашивали его глаза»). Эмоциональный шок нас интересует слабо – не стоять же, в самом деле, за спиной у пользователя, стреляя время от времени из ружья, чтобы он волновался (тем более что после шока запоминание прерывается). Достаточно и повторения с обработкой.

С повторением всё просто. Чем больше повторений и чем меньше времени проходит между повторами, тем больше шансов, что информация будет запомнена. Для нас как «людей просто» это ясно и неинтересно, но зато с точки зрения дизайна интерфейса это наблюдение вызывает очень простую эвристику: если системой придется пользоваться часто, пользователи ей обучатся, деваться-то им некуда. Это очень утешительное наблюдение.

С семантической обработкой дела обстоят интереснее. Дело в том, что информация хранится в ДВП в сильно структурированном виде (например, похоже, что зрительные воспоминания на самом деле хранятся не в виде картинки, а как список объектов, находящихся в изображении, изображения же отдельных объектов хранятся отдельно). Так что для обращения к воспоминаниям мозг выполняет работу, сходную с поиском книги в библиотеке (только более сложную; попробуйте методом самонаблюдения вспомнить, например, всех своих одноклассников). Соответственно, когда чело-

век вспоминает, он углубляется в свою память и находит всё больше признаков искомой информации. Но верно и обратное: чем больше человек думает о какой-либо информации, чем больше он соотносит её с другой информацией, уже находящейся в памяти, тем лучше он запомнит то, о чем думает[1] (т. е. текущий стимул). Это тоже очень утешительное наблюдение: если пользователь долго мучается, стараясь понять, как работает система, он запомнит её надолго, если не навсегда.

Несколько помогает понять устройство механизма запоминания его антипод, а именно забывание. Современная наука утверждает, что забывание обусловлено одним из трех факторов (или всеми тремя), а именно затуханием, интерференцией и различием ситуаций. Самое простое объяснение имеет затухание: когда информация не используется долгое время, она забывается. Несколько сложнее с двумя оставшимися факторами. Предполагается, что если сходной семантической обработке подверглись несколько фрагментов сходной информации, эти фрагменты перемешиваются в памяти, делая практически невозможным воспроизведение поврежденного фрагмента, т. е. фрагменты интерферируют друг с другом. Иначе обстоит дело с различием ситуаций. Предполагается, что для успешного воспоминания требуется соответствие признаков во время кодирования с признаками во время воспроизведения. Невозможно неслучайно вспомнить «то, не знаю что». Это всё равно как потерять книжную карточку в библиотеке – книга в целости и сохранности, но найти её нет никакой возможности.

Если серьезно, то повторение можно охарактеризовать как способ мощный, но ненадежный, поскольку трудно рассчитывать на повторение при нечастой работе с системой (существует множество систем, используемых редко или даже однократно). Семантическая же обработка есть способ мощный, но дорогой: без повода пользователи не будут задействовать свой разум, предоставить же им повод сложно. Лучше всего в качестве повода работает аналогия, неважно, как она представлена, как метафора интерфейса (см. стр. 25), или как эпитет в документации (см. «Обучающие материалы» на стр. 30).

Цена вспоминания. Является общим местом, что обращение к ДВП стоит довольно дорого. Поспорить с этим невозможно, поскольку в утверждении содержится слово «довольно», обладающее крайне размытым значением.

На самом деле всё сложно[2]. Разные понятия вспоминаются с разной скоростью, слова, например, вспоминаются быстрее цифр, а визуальные образы – быстрее слов. Очень сильно влияет объем выборки, т. е. вспомнить одно значение из десяти возможных получается быстрее, нежели из ста возможных. Наконец, частота вспоминания влияет на скорость вспоминания (т. е. на скорость вспоминания сильно влияет тренировка).

Таким образом, при проектировании интерфейса удобно пользоваться следующим правилом. Для обычных пользователей, у которых нет навыков извлечения из ДВП информации, присущей проектируемой системе, следует снижать нагрузку на ДВП; для опытных пользователей, у которых эти навыки сформировались, обращение к ДВП может быть более быстрым, нежели любой другой способ поиска информации.

Важно, однако, сознавать, что для опытных пользователей ДВП, будучи быстрым, не обязательно является предпочтительным. Например, если стоит задача снизить количество ошибок, меню (см. стр. 76) будет более эффективно, чем, скажем, командная строка, поскольку оно не позволит отдать заведомо неправильную команду.

1. Это наблюдение заставляет некоторых ученых предполагать, что реально у человека нет отдельных КВП и ДВП, различие же в функционировании памяти объясняется разной глубиной семантической обработки.
2. См. Б.М. Величковский, Современная когнитивная психология, Издательство МГУ, 1982.

Поиск и визуализация информации

Большинство задач, выполняемых с помощью компьютера, сводятся к созданию, хранению, поиску, просмотру и редактирования текстовой и численной информации, причем поиск и просмотр лидируют по затратам времени и усилий. Это делает задачу всемерного облегчения этой работы чрезвычайно важной.

Четыре вида поиска и еще два

Существует четыре основных вида поиска информации:

- Поиск конкретных данных (сколько сделок было совершено за последние два месяца?, когда родился Пушкин?)
- Поиск расширенных данных (кто еще участвовал в этой сделке, которая принесла нам столько проблем?, какие еще произведения, помимо «Мертвые душат», написал Гоголь?)
- Свободный поиск (есть ли связь между этой сделкой и какими-нибудь другими?, есть ли любовные сцены в «Кому на Руси жить хорошо»?)
- Проверка доступности (у нас есть вообще какие-нибудь данные о том, почему этот поганый контракт был подписан?, а у меня есть какие-нибудь книги Толстоевского?)

В этой таксономии есть еще один вид поиска, который я в список не включил, частично потому, что он плохо формализуется, а частично потому, что его значение до сих пор слабо осознано. Этим видом является «совсем уж свободный поиск», при котором основной вопрос, который искатель ставит перед системой, звучит как «а есть ли тут что-нибудь интересное?».

Эти четыре вида поиска (или пять) существуют очень давно, с появления первых библиотек. За сотни лет библиотекари научились очень многому, чтобы поиск нужной информации был эффективен и прост. Потом появился компьютер, вскоре придумали многомерные базы данных и королями поиска стали программисты. Проблема в том, что до сих пор информацию ищут примерно так же, как искали её сто и двести лет назад.

Основной проблемой поиска всегда было обилие информации. Нетрудно найти нужные сведения, когда у тебя всего один листок бумаги[1]. Когда же сведения нужно найти в библиотеке, состоящей из десятков и сотен тысяч (если не миллионов) листов, жизнь становится значительно более насыщенной. Для решения этой проблемы были придуманы (еще библиотекарями) картотеки, содержащие основные сведения о каждом объекте. Человек формулировал поисковый запрос, а потом тем или иным способом отбирал подходящие карточки.

Этот метод жив и поныне, хотя, конечно, в несколько других формах. Теперь это делается на компьютере (что действительно облегчило жизнь), а поиск производится языком SQL и иже с ним. Я вовсе не собираюсь вас этому учить, есть много людей, которые понимают в этом больше, чем я. Я попробую рассказать вам иное.

Дело в том, что метод карточек хорошо справляется с поиском конкретных данных и проверкой доступности. Со всеми остальными видами поиска он справляется из рук вон плохо. Возьмем, например, свободный поиск. Его цель состоит в том, чтобы найти некий паттерн, закономерность, нечто, что в начале поиска вообще неизвестно («найди то, не знаю что»). Пользуясь методом карточек, приходится совершать огромное количество поисковых запросов, держа при этом в голове полученные ранее данные. Вероятность того, что при этом будет найдена *информация, а не данные*[2], невелика.

1. Конечно, при условии, что нужная информация на этом листе вообще есть.
2. Иерархия значений собирается следующим образом: сначала идут данные, из них собирается информация (т. е. данные, отсортированные и подготовленные к использованию), из информации собирается знание (информация, лишенная несущественного), а из неё – мудрость (только общие законы без конкретики вообще).

Но выход есть. Чтобы его найти, нужно углубиться в историю и задать себе три вопроса. Вопрос первый, зачем появились карточки. Ответ: потому, что мы не в состоянии охватить взглядом всю информацию. Вопрос второй – почему в результате поиска мы вытаскиваем часть карточек из картотеки? Опять-таки потому, что мы не можем охватить их взглядом, равно как мы не можем охватить взглядом место этих карточек в общей куче. Вопрос третий – как нам оценить найденное в целом, не отвлекаясь на частности? Ответ: разложить их на столе, молясь, что стола хватит.

Всё это издержки и ограничения самой концепции поиска карточек и вытаскивания их из общей кучи. Эти ограничения имелись, пока мы не имели ничего, кроме карточек. Теперь, когда у нас есть компьютеры, эти ограничения пора отправить на свалку. Компьютер позволяет так визуализировать данные, что появляется возможность увидеть все данные (пускай издали), видя при этом в этих данных информацию.

Т. е. при таком поиске искатель не формулирует запрос, получая на выходе горсть записей базы данных, но задает правила визуализации *всех* данных и видит, какие данные либо выбиваются из общего ряда, либо наоборот слишком уж обычны. Это позволяет как найти нужные сведения, так и сразу увидеть взаимосвязи и паттерны.

При этом стандартный поиск с последовательностью запросов имеет еще один важный недостаток: он слишком абстрактен. Большинство же людей, хоть и способно создавать сложные алгоритмы, плохо управляется с абстракциями. Не имея ощутимых, не побоюсь слова «видимых», промежуточных результатов, многие люди неспособны сформировать сложный, многоступенчатый вопрос. Визуализация, напротив, позволяет это ограничение обойти.

Но не поиском единым сильна визуализация. Она позволяет также многократно сократить время, затрачиваемое на восприятие найденной информации, за счет того, что визуально выраженные закономерности воспринимаются гораздо быстрее и легче, нежели численные или цифровые (но об этом позже).

Теперь, когда я (надеюсь) убедил вас в животворящей силе визуализации данных, перейдем к практике.

Как визуализировать

Предположим, есть два числа: 83 и 332. Эти числа содержат в себе всю информацию, которую из них можно тем или иным способом вычислить (включая устройство Вселенной). Проблема состоит в том, что они никак не помогают найти нужную информацию, например, мало кто может сразу сказать, что второе число больше первого в четыре раза.

Рис. 19. Если показывать не только цифры, но и какие-либо визуальные репрезентации (слева), скорость восприятия взаимосвязи между ними многократно увеличится. Если отказаться от показа цифр (в центре), оставив только репрезентации, скорость возрастет еще больше, но потеряется возможность точно определить искомые значения (что зачастую не страшно). А если явно показать различия между значениями (справа), либо относительные, как на иллюстрации, либо абсолютные, скорость возрастет еще больше, более того, появится возможность передавать дробные значения.

Это делает числа негодными средствами для быстрой передачи их реальных значений и взаимосвязи между ними. Даже тот факт, что в десятичной системе счисления ширина числа кое-как показывает размер числа (так, понятно, что 20 меньше 200, поскольку число 200 шире[1]), не делает их лучше.

Всегда выводите цифры, предназначенные для сравнения, моноширинным шрифтом

Таким образом, отдельные числа зачастую необходимо показывать не как последовательность цифр, но как визуальные объекты, свойства которых тем или иным образом связаны с самим числом.

Несколько иная ситуация с текстовыми данными. Понятно, что более-менее сложный связный текст *автоматически* визуализировать невозможно («Мороз и солнце...»). К счастью, обычно такой необходимости и не возникает. Почти всегда нужно визуализировать сравнительные и/или описательные атрибуты, т. е. нет особой разницы между текстовыми и численными данными.

Рассмотрим пример: в гипотетической системе складского учета есть отчет, показывающий все товары на складе. Одним из вариантов вида этого отчета является банальный список. Попробуем сделать его лучше. Сначала нужно определить, зачем этот отчет нужен, и какие поля должны быть в списке. Здесь очень многое зависит от ситуации, поэтому я не буду производить попытки таксономии складов, а просто скажу: предположим, что нам нужно облагородить столбцы Товар, Дней до порчи, Количество и Занимаемый объем. Поехали.

Без каких либо усилий, только лишь благодаря запросу, получится примерно такой список (предположим, что складское место измеряется в квадратных метрах стеллажей):

Товар	Дней до порчи	Количество	Объем	..
Ночной горшок	---	1000 шт.	3	..
Говядина	26	12 тонн	15	..
Детектив «Иван – человек с ружьём»	---	2000 шт.	4	..
Зубная паста «Желтый жемчуг»	142	80 коробок	5	..
Партайгеноссе Путин	увы, немало	Один	1	

Рис. 20. Начальное состояние отчета.

Это явно можно сделать лучше[2]. Начнем со второго столбца – дни до порчи. Как видим, отображаются числа, которые, как мы уже знаем, воспринимаются «не так чтоб очень». Но прежде чем что-либо менять, нужно определить, зачем вообще этот столбец нужен. Судя по всему, его предназначение заключается в том, чтобы сделать ненужным отчет «Товары, которые вот-вот придется выкидывать». Из этого можно сделать два вывода:

- Точное число дней или месяцев не так уж и важно. Важно, много времени осталось до порчи или нет.
- Помимо знания времени, оставшегося до порчи, важно так же сознавать, насколько товар уже попорчен, т. е. какую часть общего срока хранения составляет остаток.

1. Это опасно тем, что трудно оценить малые различия между числами на границе порядков (99 *не сильно* меньше 100) и в тех случаях, когда числа не целые (число 12.347 меньше числа 13).
2. Вообще говоря, эту таблицу можно сделать лучше и без всякой визуализации. Например, второй столбец, выровненный по правому краю, будет сканироваться взглядом гораздо быстрее. Но писать об этом я не хочу, поскольку эта тема подробно рассмотрена в литературе по коммуникационному дизайну.

Эти выводы подразумевают множество возможных решений. Выбор одного варианта в таких условиях обычно затруднен, поскольку на него влияет много факторов и поиск компромисса довольно сложен. Однако пойдем по порядку.

Пункт первый, количество оставшихся дней. В таблице из примера есть определенные проблемы – имеющаяся нотация никак не учитывает того, что измерять большие промежутки времени удобнее в неделях и месяцах. В примере же спокойно написано «142 дня». С одной стороны это хорошо, потому что сразу видно, что времени осталось много. С другой стороны, трудно понять, *сколько именно* осталось времени. Т. е. лучше было бы писать либо «4 мес 22 дн», либо «4 мес 3 нед 1 дн», но, с ещё одной стороны, громоздкость таких конструкций не позволяет надеяться, что их восприятие будет легким и быстрым (не надо забывать и о существовании годов).

При визуализации массивов отдельных параметров важно добиться не просто красоты и понятности отдельных блоков, но легкости прочтения многих блоков за малое время. Популярно говоря, каждый вариант проверяйте на большом количестве данных, не ограничивайтесь проверкой на одном значении.

Тут возможно несколько способов улучшения. Первый и самый эффективный способ, а именно представление чисел как визуальных объектов, здесь может и не пройти: для этого способа может потребоваться либо слишком много места на экране (которого нет), либо слишком много цветов (из-за которых экран станет похож на новогоднюю елку и перестанет быть читаемым). Но попытка не пытка: за пятнадцать минут я изготовил пару вариантов решения.

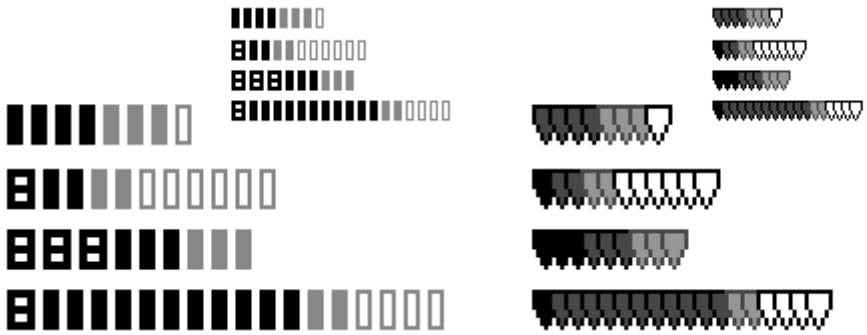


Рис. 21. Два варианта визуализации времени, оставшегося до протухания товара. Единицы времени упорядочены по продолжительности, т. е. самые темные блоки обозначают годы (к сожалению, в первом варианте обойтись только изменением насыщенности не удалось), а самые светлые, располагающиеся справа – дни. Первый вариант более разборчив, зато второй занимает меньше места. Если бы можно было использовать цветовое, а не только «световое» кодирование, разборчивость можно было сильно повысить.

Оба варианта эксплуатируют одну и ту же идею: чем длиннее и темнее полоска, тем больше осталось времени. Конечно, для точного определения оставшегося времени они не слишком удобны, но, как уже было сказано, это обычно и не нужно, тем более что численное значение всегда можно выводить, например, в контекстной всплывающей подсказке.

Разумеется, эти варианты имеют еще и тот недостаток, что они занимают много места на экране: что будет, если понадобится отобразить продолжительность в десять лет, 11 месяцев, три недели и семь дней? Здесь, однако, тоже есть решение: чем больше срок, тем менее актуальна точность его отображения, например, совершенно неважно, сколько дней может пролежать на складе мыло, если срок его хранения все равно измеряется годами. Это позволяет отказаться от показа частностей, т. е. можно

установить правило, согласно которому у товаров, которые могут хранить больше года, не показываются недели и дни. Также можно показывать короткие полоски разреженными, а длинные – уплотненными. Не говоря уж о том, что можно воспользоваться обоими способами одновременно.

Конечно, в некоторых случаях такой способ вывода чисел неприемлем и приходится от чего-то отказываться. Тем не менее, даже и отказавшись от многого, можно облегчить восприятие этих чисел: можно, например, визуально отображать годы, месяцы и дни, оставляя числам собственно значения (см. рис. 22).

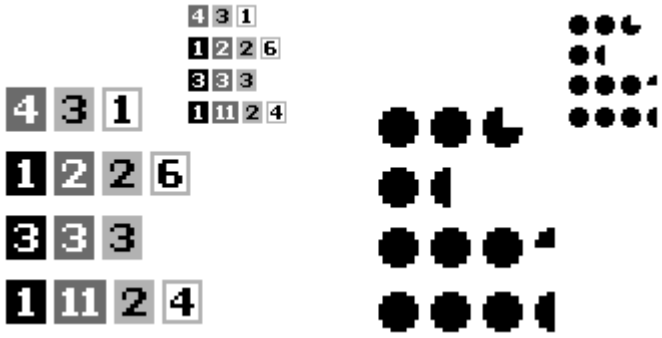


Рис. 22. Ещё два способа визуализации. Первый вариант визуально отображает годы, месяцы, недели и дни, оставляя числам собственно передачу значения, что частично позволяет воспользоваться идеей связи длины и светлоты полоски с реальным численным значением. С другой стороны, числа на полосках отвлекают, а значит, замедляют распознавание. Во втором варианте отображаются только одни единицы отсчета времени, благодаря чему достигается самая большая легкость и скорость распознавания, равно как сохраняется приемлемый размер блока. Другой разговор, что такое решение может работать только в том случае, когда все товары, которые хранятся на складе, имеют длительности хранения одного порядка, измеряемые, например, в неделях.

Теперь о втором пункте. Важно сознавать, насколько товар уже попорчен. Для этого есть множество стандартных индикаторов, из которых наиболее удобными являются линейный и круговой.

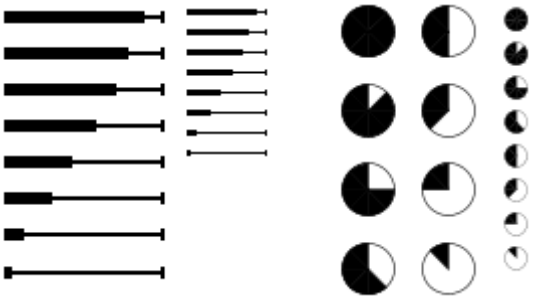


Рис. 23. Индикатор оставшейся доли. Левый вариант плох тем, что занимает довольно много места, правый меньше, зато его легче прочесть обратным образом, т. е. неправильно. С другой стороны, с правым вариантом легче индицировать критический случай, т. е. ситуацию, при которой товар вот-вот испортится: надо лишь покрасить его в красный цвет.

Выбор варианта обычно сильно зависит от контекста. Поскольку пространство экрана обычно весьма дорого, обычно выбирают меньший вариант, но это не догма. Чтобы избежать путаницы, эффективнее всего выбрать тот вариант, который не используется в том же отчете, при этом сам вид индикатора будет подсказывать пользователю, на какой именно столбец он смотрит.

Идеально было бы придумать обобщенный индикатор, показывающий оба смысла. К сожалению, чаще всего (и в данном случае в частности) это невозможно, поскольку в одном индикаторе пришлось бы показывать слишком большой объем информации. Тем не менее, прилагать усилия к достижению этого идеала стоит вполне.

Итого, со столбцом «Дней до порчи» разобрались. Пора обновить отчет. Предположим, что наиболее подходящим индикатором времени, оставшегося до порчи товара, была признана полоска с треугольниками, а индикатор оставшейся доли срока хранения было решено показывать только тогда, когда этой доли осталось менее половины. Тогда отчет станет выглядеть примерно так:

Товар	Дней до порчи	Количество	Объем	..
Ночной горшок		1000 шт.	3	..
Говядина		12 тонн	15	..
Детектив «Иван – человек с ружьём»		2000 шт.	4	..
Зубная паста «Желтый жемчуг»		80 коробок	5	..
Партайгеноссе Путин	увы, немало	Один	1	

Рис. 24. Промежуточное состояние отчета. Улучшен только второй столбец.

Итого, со столбцом «Дней до порчи» разобрались. Следующим идет столбец «Количество». Здесь другая проблема – единицы измерения у разных товаров различаются, что не позволяет визуализировать «сравнительность» параметров. Невозможно также визуализировать количество элементов, поскольку это количество может быть очень большим (визуализировать тысячу ночных горшков можно, но при этом будет затрачено чуть ли не всё пространство экрана, а главное, явной разницы между девятью сотнями и тысячей горшков видно не будет). Таким образом, остается лишь возможность визуализации единиц измерения, в данном случае штукам, тоннам и коробкам можно нарисовать пиктограммы.

Весь вопрос в том, стоит это делать или нет. Большинство людей получает такое удовольствие от рисования пиктограмм, что сам вопрос необходимости этого рисования отходит на второй план. В то же время пиктограммы в такой роли имеют как достоинства, так и недостатки. Недостатки у пиктограмм следующие (более подробно о пиктограммах см. стр. 100):

- им нужно учиться
- если разные пиктограммы похожи друг на друга, количество случаев неправильного распознавания неприлично возрастает
- они добавляют лишний визуальный шум
- на их рисование нужно время
- при появлении необходимости в еще одной пиктограмме возникает много проблем.

С другой стороны, у пиктограмм есть и достоинства: разборчивость и эстетическая насыщенность. Проблема в том, что если недостатки совершенно реальны, то достоинства потенциальны: стоит эти пиктограммы нарисовать *не очень хорошо*, как разборчивость и красота начисто исчезают. Так что пользоваться пиктограммами, как индикаторами в списках, чаще всего неэффективно. Однако, если точно известно, что количество единиц измерения невелико, а найти их внятные репрезентации непроблематично, пиктограммы могут быть более чем хороши. Но случается это реже, чем хотелось бы.

Так что идею рисования пиктограмм для этого столбца мы оставим кому-нибудь другому. Следующий столбец – «Объем». С ним с одной стороны легко, с другой – не очень. Единица измерения всего одна, что хорошо. Зато объем, занимаемый товаром, может быть большим, а значит, плохо визуализирующимся (и пятнадцать метров стеллажей говядины визуализировать нелегко, что уж говорить о ста метрах). Но это всё техника. Главное, напротив, не визуализировать параметр, а визуализировать нужную информацию из этого параметра.

И с этим-то как раз не всё ясно. Что пользователь хочет узнать из чисел, показывающих, сколько места на складе занимает товар? Если ему нужно всего лишь узнать, много он занимает места или нет, лучше всего будет работать уже описанный индикатор-полоска. Если ему нужно точно узнать, сколько места занимает товар, можно оставить числа. Но вполне может оказаться и так, что ему нужно узнать, сколько процентов общей площади склада занято товаром. В этом случае подойдет либо полоска, либо более компактная круговая (блочная) диаграмма, либо те же числа. Основная проблема визуализации заключается именно в определении потребностей пользователей.

Предположим, что нужен как раз процент. Поскольку малые проценты показываются на малюсенькой диаграмме не очень хорошо (равно как и на полоске), числа лучше оставить, но сделать их поддерживающими. Подразумевается, что если пользователю нужно грубое, но быстрое знание, он смотрит на диаграмму. Если ему нужна дополнительная точность, он переводит взгляд на число. А чтобы число не отвлекало без нужды, лучше снизить его визуальную насыщенность, проще говоря, выводить число процентов не черным, а серым цветом. Единственно надо обратить внимание, что большинство товаров будут занимать на складе сравнительно небольшое место. Это значит, что индикатор надо оптимизировать не для показа всех возможных значений, а для показа их части (при этом части самой неудобной для показа). Например, так:

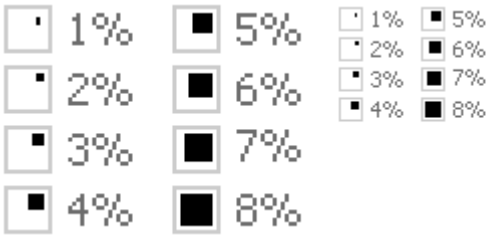


Рис. 25. Вариант вывода процентных значений. Обратите внимание, что этот вариант плох отсутствием универсальности: пользуясь им, нельзя показать значения, большие 8% (графику придется заменять голыми цифрами). С другой стороны, с практической точки зрения это может быть и не страшно – не так уж много складов, которые до отказа забиты чем-то одним.

Обратите внимание, что процентное значение вполне может оказаться дробным. В нашем случае от идеи визуализации процента придется отказаться. Искомый отчет в качестве полигона графического показа процентов не подходит: говядина занимает пятнадцать складских мест, а ночные горшки всего три, причем есть товары и на четыре и на пять мест. Установить процентные соотношения в таких условиях возможно, но вероятность того, что эти проценты будут «слишком уж дробные» (например, 0.273), чтобы их можно было визуализировать, слишком велика. Не надо забывать, что в нашей таблице только четыре строки, а в жизни строк может быть и двести и триста. Кто знает, какие параметры там могут оказаться? Но показ процентов, в общем-то, экзотичен. Почти всегда нужно просто показывать, большое значение или, наоборот, маленькое. В таких условиях обычная полоска из блоков справляется на ура.

Таким образом, в результате отчет может выглядеть примерно так:

Товар	Дней до порчи	Количество	Объем	..
Ночной горшок		1000 шт.	▽▽▽	..
Говядина	▓▓▓▽	12 тонн	▓▓▓▓▓▓▓▓▓▓▓▓	..
Детектив «Иван – человек с ружьём»		2000 шт.	▽▽▽▽	..
Зубная паста «Желтый жемчуг»	▓▓▓▓▽	80 коробок	▽▽▽▽▽	..
Партайгеноссе Путин	увы, немало	Один	1	

Рис. 26. Финальное состояние отчета.

Что гораздо лучше, поскольку более разборчиво и, пожалуй, эстетичней.

Осталось сказать одно: эта глава в большей степени демонстрирует алгоритм мышления, нежели рекомендует какие-либо конкретные действия. Я совершенно уверен, что при достаточном прилежании вам удастся изобрести значительно лучшие индикаторы, равно как и использовать существующие лучше, нежели это получается у меня. Здесь всё в ваших руках. И ещё. Поскольку большая сила подразумевает большую ответственность, никогда не пытайтесь визуализировать то, что визуализировать не надо. Ни к чему хорошему это не приведет.

Теперь о больших объемах данных. Поскольку эта тема более сложна и менее нужна (точнее – реже нужна в интерфейсе), нежели визуализация малого объема, сюда вместилось только краткое введение в проблему.

Визуализация большого объема данных сильно отличается от визуализации малого объема. Если при малом объеме данных нужно показывать сами эти данные, то при большом объеме нужно показывать *распределение* значений. С одной стороны, это проще, поскольку самих данных нет, есть только распределение. С другой стороны, каждому критерию необходима либо своя ось, либо иной метод выделения.

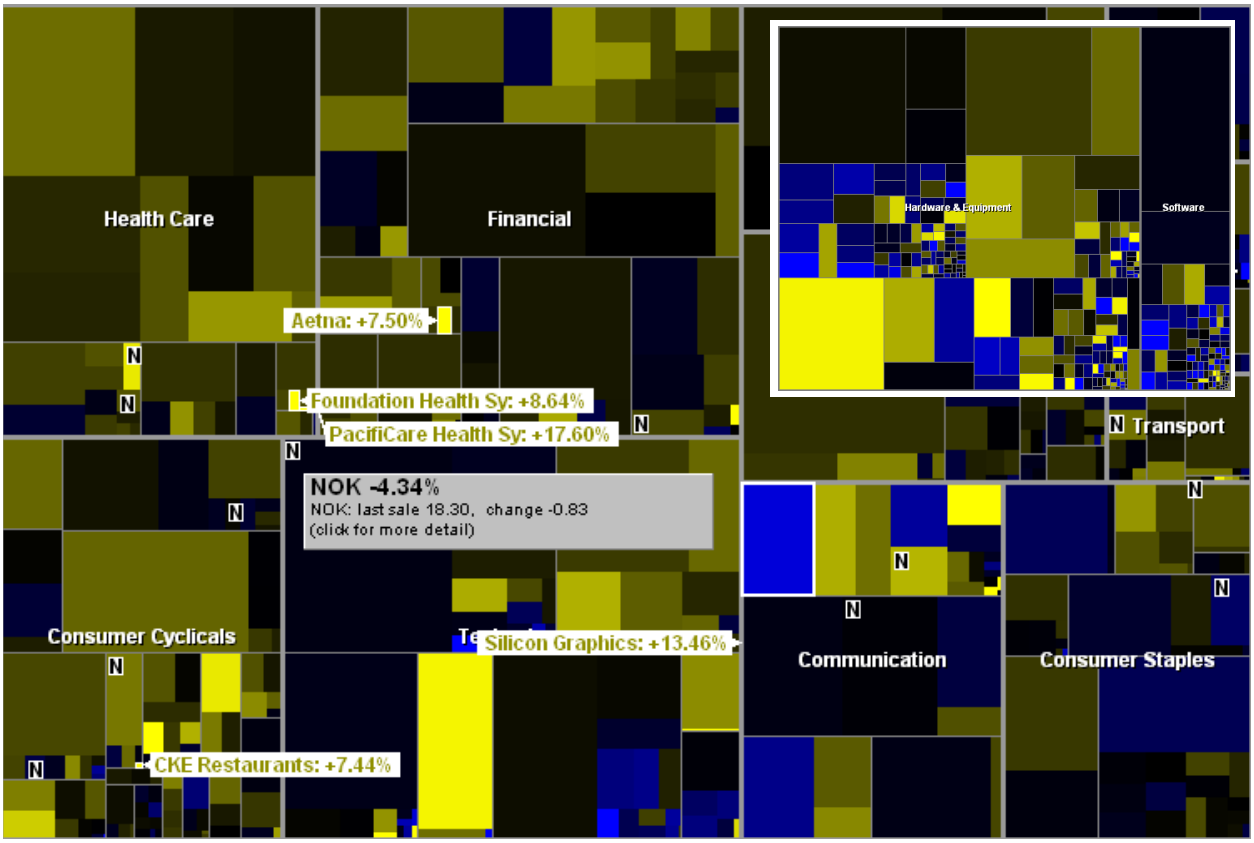


Рис. 27. Пример крайне удачной системы визуализации данных. Эта карта показывает состояние торгов на бирже NYSE. Карта поделена на зоны, такие как Здравоохранение и Энергетика. Каждый прямоугольник показывает состояние акции одного эмитента. Размер прямоугольника показывает объем торгов конкретными акциями за текущий день, цвет – изменение стоимости акций (синие теряют в стоимости, желтые приобретают). © SmartMoney http://www.smartmoney.com/maps/

Если осей немного, всё просто. Распределение эффективнее всего строить в виде либо двумерной карты, либо в виде псевдо-трехмерного пространства. Как правило, трехмерность не очень эффективна, поскольку она требует от пользователя долго искать наиболее удобный ракурс (более того, чтобы увидеть эту трехмерность, карту нужно некоторое время

вращать). С другой стороны, эти ненужные по сути дела действия могут приносить пользователям удовольствие, так что потеря времени будет скомпенсирована.

Читателям, заинтересовавшимся этой темой, настоятельно рекомендую, во-первых, прочесть уже упоминавшиеся книги Эдварда Тафта, во-вторых, почерпнуть вдохновение в любом пакете математического анализа и статистики (например, Mathematica или Mathcad).

Навигация

В любой системе, использующей единое устройство вывода (читай – экран) для любых задач, качество навигации является важной составляющей: поскольку все физические атрибуты системы фактически отсутствуют, их необходимо передавать визуалом экрана. Наличие в системе развитой навигации здорово облегчает процесс обучения работе с системой, поскольку при визуальной навигационной системе не нужно помнить контекст своих действий (как я сюда попал? что я хотел сделать?), поскольку этот контекст сам по себе показывается на экране.

Разработка навигационной системы есть сложная и большая работа, требующая большого количества знаний, навыков и талантов. К сожалению, эта книга не способна дать вам ни первого, ни второго, ни третьего, тем более что тема навигации хорошо проработана в литературе. Тем не менее, здесь всё-таки рассматриваются две темы: первая слишком часто забывается, а вторая недостаточно хорошо проработана в существующей литературе.

О навигации в пределах документа см. «Полосы прокрутки и их альтернатива» на стр. 91.

Цели навигации

Любая система навигации обязана выполнять несколько функций, а именно показывать пользователям:

- **Где они находятся сейчас.** Для понятия «не знать своё теперешнее положение» есть два коротких синонима: заблудиться и потеряться. Ни то, ни другое не приносит удовольствия.
- **Куда они вообще могут переместиться.** Невозможность узнать, что можно сделать дальше, приводит к тому, что ничего не делается.
- **Где они уже побывали.** Невозможность определить пройденный маршрут приводит к тому, что пользователи попадают туда, куда попасть они не хотят (они тут уже были), что является человеческой ошибкой.
- **Куда им разумно пойти.** В большинстве случаев пользователям удобнее не думать самим о том, что им нужно сделать, но воспользоваться готовым вариантом действия, а сбереженные ресурсы потратить на что-либо иное.
- **С какого именно экрана (страницы) они пришли.** Точно знать направление своего движения полезно, поскольку это помогает поддерживать контекст действий.

При этом от навигационной системы требуется множество дополнительных свойств, таких как эстетическая привлекательность, малый размер и так далее (эти свойства не вполне интерфейсные, поэтому здесь они не разбираются).

Так вот, очень важно сделать навигационную систему, которая выполняет максимум этих задач, просто потому, что невыполнение части из них делает систему неполноценной. Всякий раз, создавая навигационную систему, нужно вкладывать в неё максимальное количество ответов.

Битва против меню

Почти любая навигационная система является меню. Это имеет множество достоинств, но также и некоторые недостатки.

Во-первых, меню ограничивает возможность выбора только определенным набором вариантов. Это хорошо, поскольку позволяет исключить заведомо неправильные действия. Но в навигационных системах это оказывается нехорошо: взамен свободного передвижения по системе

пользователям приходится ходить по заранее проделанным для них тропинкам, т. е. степень свободы пользователей сокращается. Иногда это полезно, но чаще – нет.

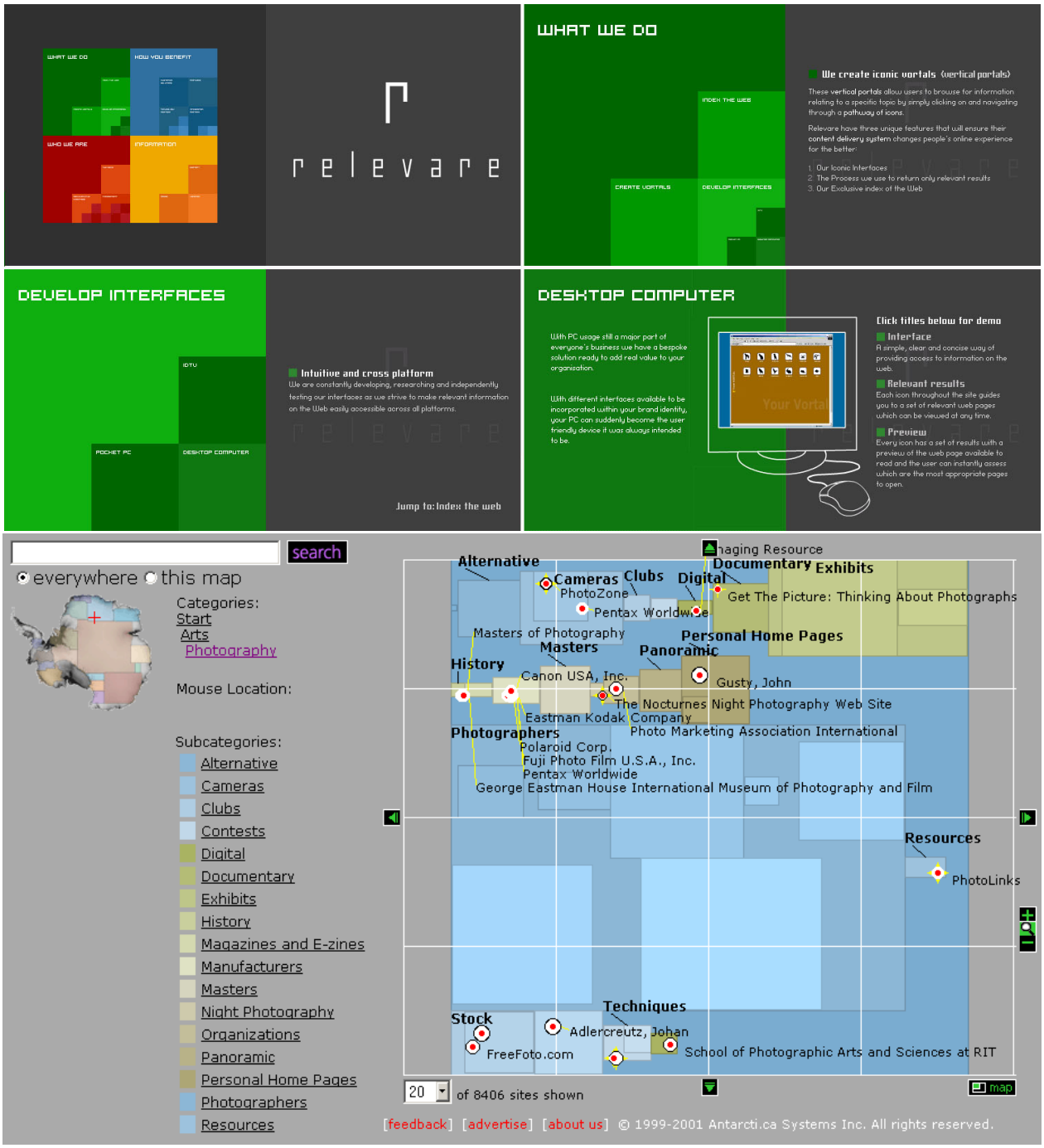


Рис. 28. Примеры навигационных систем, которые принципиально действуют одинаково, но выглядят различно и выполняют разные цели. Сверху приведены четыре последовательных экрана сайта фирмы Relevare (читаются слева направо), в которых навигационная система жестко задана один раз. Снизу экран системы Antarcti.ca, позволяющей перемещаться по всем сайтам из каталога Yahoo. Единые принципы устройства навигационных систем могут быть применены многими разными путями, главное – провести анализ требований к навигационной системе и подойти к делу творчески. © Relevare, © Antarcti.ca

Во-вторых, любое многоуровневое меню страдает от каскадных ошибок (см. стр. 80). Способы борьбы с ними существуют, но от этих ошибок лучше бы вообще избавиться. Сделать же это можно, только изъяв либо само меню, либо потребность его использовать.

В-третьих, большинство меню не способно показать пользователям, куда им разумно пойти. Это не значит, что меню плохо, но только то, что нужно как-то усиливать навигацию помимо меню.

Единственным универсальным (и работоспособным) алгоритмом решения всех этих проблем является создание системы из двух не связанных между собой навигационных систем. Первая (меню) показывает пользователю, где он находится и куда он может перейти. Вторая, присутствующая, возможно, не на каждом экране, показывает пользователю, куда он, вероятнее всего, хочет перейти в данный момент. Впервые такая система добилась известности на сайте Amazon.com (хотя идея изначально появилась на firefly.com, к сожалению, этот сайт прекратил существование). Система анализирует читательские предпочтения, проще говоря, регистрирует, кто что купил и на какие товары перед этим смотрел. Собрав эти данные, система предлагает их пользователям, т. е. позволяет просматривать товары не по абстрактным группам, но по покупательским предпочтениям. Никто, однако, не мешает не собирать информацию, но устанавливать связи, пользуясь результатами анализа действий пользователя и здравым смыслом. При написании книг это делается уже порядка сотни лет: блок «См. также» в конце главы или раздела есть не что иное, как дополнительная система навигации по книге.

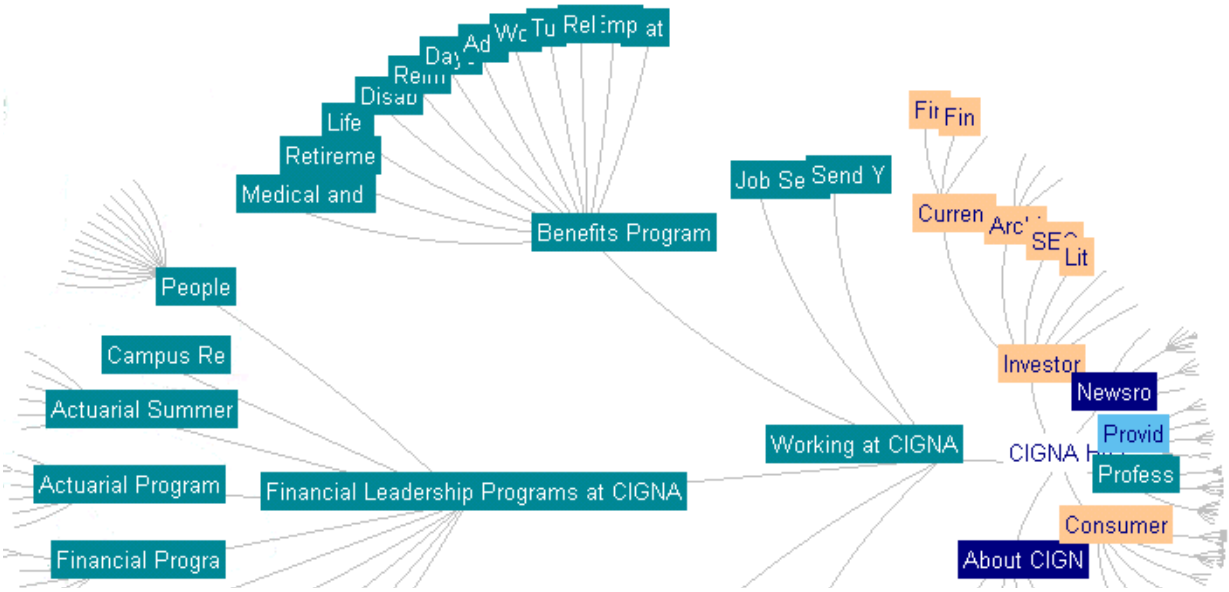


Рис. 29. Это не выглядит как меню, это не воспринимается как меню. Тем не менее, по содержанию это обычное меню, ничем не отличающееся от меню абсолютного большинства сайтов. С другой стороны, такая форма представления меню позволила резко увеличить объем структуры, вмещающейся на экран, и тем самым упростить поиск нужных фрагментов системы (при этом субъективное удовлетворение здорово повышается). © Inxight Software, http://www.inxight.com

При этом меню становится не основным, а дополнительным навигационным инструментом (точнее, пассивным инструментом: на него можно смотреть, чтобы получить ответы на интересующие вопросы, но с ним не обязательно взаимодействовать).

Часть ii

Инвентарь

В этой части описаны все кирпичи, из которых состоит любая программа или сайт. Это окна, элементы управления и стандартные сценарии поведения системы.

Элементы управления

Начать разговор об элементах управления удобно с самого простого, а именно с кнопок.

Кнопки

Кнопкой называется элемент управления, всё взаимодействие пользователя с которым ограничивается одним действием – нажатием. Эта формулировка, кажущаяся бесполезной и примитивной, на самом деле очень 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).

С точки зрения разработчика ПО для настольных систем, командные кнопки являются чрезвычайно простыми и скучными. Иная ситуация в интернете, где отсутствие операционной системы (откуда приходят элементы управления) и простота создания новых типов кнопок (это чуть ли не единственный элемент, с которым вообще удается что-либо сделать) привели к тому, что нестандартные кнопки не создает только ленивый. В то же время, этот самый простой элемент управления имеет больше всего тонкостей.

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. Как вы уже знаете, чем больше кнопка, тем легче попасть в нее курсором. Это правило по мере сил всеми соблюдается, во всяком случае, кнопок размером 5 на 5 пикселей уже практически не встретишь. Однако помимо простоты нажатия на кнопку есть другая составляющая проблемы: 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. Первый способ пробрел существенную популярность в интернете, второй – в обычном ПО (он, кстати, более эффективен: одно дело, когда пользователь промахивается мимо кнопки и совсем другое – если, промахнувшись, он ещё и ошибочно нажимает на другую кнопку). Ни тот, ни другой способы не обеспечивают стопроцентной надежности, так что при прочих равных использовать стоит оба.

Считать экранную кнопку нажатой нужно не тогда, когда пользователь нажимает кнопку мыши, а курсор находится на кнопке, а тогда, когда пользователь отпускает нажатую кнопку мыши, курсор находится на экранной кнопке и находился на ней, еще когда кнопка мыши нажималась

Другой составляющей проблемы размера кнопок в интернете является несоответствие видимой площади кнопки её действующей площади. В последнее время, кнопки часто реализуют посредством окрашенных ячеек таблицы, в которых размещается текст, являющийся гипертекстовой ссылкой. Проблема заключается в том, что пользователи воспринимают кнопкой всю ячейку, хотя реально «нажимается» лишь малая её часть.

Рис. 31. Пример несоответствия видимой области кнопки её активной области. В примере слева кнопка нажимаема, когда курсор находится в пределах текста, но не нажимаема, когда он находится в пределах кнопки, но не в пределах текста. Справа правильный вариант.

Объем. Кнопка должна (или не должна) быть пользователем нажата. Соответственно, пользователю нужно как-то сигнализировать, что кнопка нажимаема. Лучшим способом такой индикации является придание кнопке псевдообъема, т. е. визуальной высоты. С другой стороны, этот объем плох тем, что при его использовании возникает рассогласование между обликами кнопок прямого и непрямого действия. Разумеется, никто не отменял ещё и тот факт, что псевдообъем кнопок, вообще говоря, в существенной степени есть визуальный шум. Еще с одной стороны, зачастую возникает необходимость максимально повышать шансы нажатия пользователем какой-либо отдельной кнопки (например, «О компании»), в этих случаях псевдообъем этой кнопки (при прочих плоских) сильно повышает вероятность нажатия.

Направление теней во всех элементах управления должно быть одинаковым: снизу справа

Состояния. Кнопка должна как-то показывать пользователям свои возможные и текущие состояния. Количество состояний довольно велико, при этом наборы возможных состояний в ПО и в интернете значительно различаются. Например, кнопка в Windows может иметь шесть состояний: нейтральное, нажатое, нейтральное с установленным фокусом ввода, состояние кнопки по умолчанию, кнопка по умолчанию с установленным фокусом ввода и заблокированное состояние (см. рис. 32). В интернете обычно используют меньший набор состояний: нейтральное, готовое к нажатию (onMouseOver) и активное (в случаях, когда набор кнопок используется для индикации навигации). Нажатое и заблокированное состояние используются очень редко, а «нейтральное с установленным фокусом ввода» старается, как может, создать браузер.

Рис. 32. Состояния кнопки в Windows: нейтральное, нажатое, нейтральное с установленным фокусом ввода, состояние кнопки по умолчанию, по умолчанию с установленным фокусом ввода и заблокированное.

Вообще говоря, обычно, чем больше набор состояний, тем лучше. Но главное не это, а отсутствие дублирования состояний: не должно быть разных состояний, выглядящих одинаково. Также очень важно делать заблокированные состояния действительно заблокированными: так, например, в интернете очень часто встречаются кнопки, нажатие на которые открывают ту же самую страницу, т. е. нажатие которых возможно, но бесполезно. Такие кнопки должны не только выглядеть заблокированными (менее яркими и значительными, нежели обычные), но и не нести гипертекстовых ссылок.

Никогда не удаляйте элементы, которые нельзя нажать, взамен этого делайте их заблокированными

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). Разработчики же интерфейса с не менее изумительным упорством не следуют этому правилу. Аргументов у них два: во-первых, все так делают, значит, это есть стандарт и ему нужно следовать, во-вторых, нет времени придумывать название.

Оба аргументы сильны. Действительно, стандарт. Действительно, нет времени. Но есть два контраргумента: во-первых, это не столько стандарт, сколько стандартная ошибка, во-вторых, думать можно и по дороге домой. Если второй контраргумент особых объяснений не требует, то сущность первого полезно объяснить.

Кнопка, запускающая действие, недаром называется командной. С её помощью пользователи отдают системе команды. Команда же в русском языке формируется посредством глагола в повелительном наклонении (никто особо не хочет слишком персонифицировать компьютер и обращаться к нему в наклонении просительном, т. е. Придите, Увидьте, Победите).

Помимо этого, у глагольных кнопок есть одно большое достоинство. По ним понятно, какое действие произойдет после нажатия. Это позволяет как-то разграничить диалоговые окна в сознании (поскольку разные диалоговые окна получают разные кнопки). В результате, из-за увеличения степени уникальности фрагментов системы, обучаться системе получается лучше, нежели с кнопками, одинаковыми везде. Более того, вкупе со строкой заголовка окна, глагольные кнопки создают контекст, что очень полезно при возвращении к прерванной работе. Оказывается возможным не рассматривать *всё* диалоговое окно, чтобы узнать, на каком действии задача была прервана – достаточно просто прочесть надпись на кнопке. Таким образом, следует всемерно избегать создания кнопок с ничего не говорящим текстом, поскольку такой текст не сообщает пользователям, что именно произойдет после нажатия кнопки.

При этом есть одна тонкость. Существующие интерфейсы заполнены терминационными кнопками Ок, Отмена (Cancel) и Применить (Apply), что, собственно говоря, и позволяет разработчикам ссылаться на стандарт. Эти кнопки плохи.

С первой кнопкой понятно – это не глагол, а значит, кнопка плоха. Кнопка одна и та же во всех диалогах, значит всё ещё хуже[1].

1. У кнопки ОК, с другой стороны, есть два достоинства. Во-первых, слишком часто оказывается, что ОК является единственным текстом, который можно уместить в кнопке (если во всех отношениях адекватный глагол слишком длинный). Во-вторых, стандартность этой кнопки приводит к почти мгновенному её распознаванию, что позволяет ускорить работу пользователя с системой (с другой стороны, это распознавание может быть неверным, так что появляется риск человеческой ошибки).

Вторая, хоть и почти глагол, плоха, поскольку не дает контекста (к тому же отглагольные существительные воспринимаются медленнее, чем соответствующие глаголы). Главный её недостаток, впрочем, заключается в том, что её, как правило, нечем заменить, так что приходится пользоваться ею.

Третья, будучи и глагольной, и сравнительно уникальной, имеет другой недостаток: она почти всегда используется неправильно. На ней написано Применить, но на самом деле её значение совсем иное. Разберем это подробнее.

Как правило, разработчики создают диалоговое окно, внизу которого располагают три кнопки: Ок, Применить и Отмена (прямо-таки триединство ошибки). Проблемы наступают тогда, когда пользователь делает что-либо в диалоговом окне и начинает думать, какую кнопку ему нужно нажать. Предположим, он всем доволен и нажимает кнопку ОК. Не считая слабо переданного контекста, все довольно хорошо. Все довольно неплохо, если пользователь нажмет кнопку Отмена – его команды просто не будут обработаны системой.

А теперь предположим, что пользователь нажал кнопку Применить. Система выполняет команду пользователя и меняет данные. Начинается самое интересное: теперь кнопка ОК не делает ничего (команда-то уже обработана), помимо закрытия окна. Т. е. эту кнопку в данном состоянии нужно переименовывать в Закрыть. Более того. Кнопка Отмена после нажатия кнопки Применить тоже начинает врать пользователю: она не отменяет действие, но просто закрывает окно. Таким образом, если делать интерфейс полностью однозначным, получается гадость: последовательность кнопок Ок, Применить и Отмена после нажатия кнопки Применить превращается в последовательность Закрыть, Применить, Закрыть.

Помимо того, что это просто глупо, это плохо уже и тем, что пользователь оказывается обманут: он-то думает, что если он нажмет кнопку Отмена, его действия в диалоговом окне не будут приняты системой во внимание. В результате, если пользователь нажмет сначала кнопку Применить, а потом кнопку Отмена, он гарантированно совершит ошибку, в которой виновата система.

Напротив, если бы вместо кнопки Применить была бы кнопка Предварительный просмотр, все бы работало великолепно. Мало того, что пользователь не путался бы в кнопках, он мог бы избежать многих ошибок, просмотрев результат своих действий перед их окончательным принятием. Но разработчикам реализовывать режим предварительного просмотра тяжело[1]. Гораздо легче вставить кнопку Применить, а то, что пользователям это вредно, их не касается.

Таким образом, кнопка Применить оказывается не просто ненужной, но и откровенно вредной. Её можно применять только в палитрах, заменяя ею кнопку ОК, чтобы показывать пользователю, что палитра не исчезнет с экрана после нажатия кнопки. Разумеется, в этом случае с нею должна использоваться кнопка Закрыть (вместо кнопки Отмена). Во всех остальных случаях кнопка Применить не нужна.

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 (об этом см. «Пиктограммы» на стр. 100). Конечно, даже и нераспознанная пиктограмма хороша тем, что она визуально отделяет кнопку от кнопки и для опытных пользователей обеспечивает ускорение при поиске нужной кнопки (пользователь может помнить, что ему нужна кнопка с синим пятном на пиктограмме). Так что, судя по всему, 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

1. Также огорчительно то, что название *Предварительный просмотр* чрезвычайно велико и редко помещается на кнопке.

различие 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

Также к группе командных кнопок относится кнопка доступа к меню. Формально, это попытка скрестить ужа (раскрывающийся список, см. стр. 71) с ежом (кнопкой, см. стр. 63), но попытка удачная.

Идея проста. 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. В самой этой мысли нет ничего плохого, но.

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. Формально, для совсем уж неопытных пользователей, кнопка работать будет, но, как только пользователи подрастут, контекстное меню окажется эффективней. Как никак, в кнопку надо попасть курсором, а в меню попадать не надо, достаточно просто нажать правую кнопку мыши.

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.

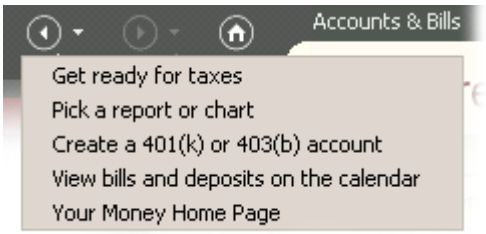


Рис. 33. Пример очень хорошего использования кнопки доступа к меню. Большинство пользователей в большинстве случаев не думают ни о каком меню, сразу нажимая на кнопку и инициируя действие. В то же время они получают возможность без труда сэкономить несколько нажатий, если обратятся к меню.

Существуют, впрочем, 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. Другой пример: иногда бывают ситуации, когда действий может выполняться несколько, но чаще всего нужно только одно. В этом случае пользователи очень быстро обучаются этому действию, имея довольно простой доступ к остальным. В таком исполнении кнопки доступа к меню работают замечательно.

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.

Отдельного внимания заслуживает размещение кнопок в пределах диалоговых окон, об этом см. «Структура окна» на стр. 93.

Несмотря на то, что в большей части литературы чекбоксы и радиокнопки разбирают по отдельности, я решил нарушить эту традицию и описать их вместе, поскольку это помогает лучше понять различия между этими элементами.

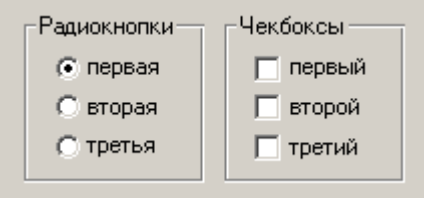


Рис. 34. Пример радиокнопок и чекбоксов. Обратите внимание, что радиокнопки *всегда* должны находиться в рамке группировки[1], а для чекбоксов это необязательно.

Первое, что необходимо сказать про чекбоксы и радиокнопки, это то, что они являются кнопками отложенного действия, т. е. их нажатие 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 опасно, 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 даёт возможность пользователям выбрать любую комбинацию параметров, радиокнопки же позволяют выбрать только один параметр. Это сближает эти элементы со списками множественного и единственного выбора соответственно (о которых будет подробнее рассказано ниже).

Из этого различия проистекают все остальные. Например, в группе не может быть меньше двух радиокнопок (как можно выбрать что-либо одно из чего-либо одного?). Еще одно следствие заключается в том, что 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). Но это было знание вообще. Теперь перейдем к алгоритму его использования.

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. Дело в том, что группу из двух радиокнопок часто можно заменить одним чекбоксом. Предположим, что нужно дать пользователю выбор: показывать в документе линейки или не показывать. В этом случае логично поместить в диалоговое окно рамку группировки (см. «Структура окна» на стр. 93) со словами Показывать линейки, а в эту рамку поместить две радиокнопки: Да и Нет. Понятно, что это решение очень тяжеловесно. Можно сделать проще: убрать рамку группировки и радиокнопки, а на их место поместить всего один чекбокс со словами Показывать линейки. В этом случае все будет хорошо. К сожалению, этот метод работает не всегда. Поскольку в самом чекбоксе

1. Или быть единственными элементами управления в окне.

написано только то, что произойдет после его включения, но не описано, что произойдет, если его не включить, такая конструкция не работает в ситуациях, когда пользователям по той или иной причине функциональность непоставленного чекбокса может быть непонятна. Например, если нужно спросить пользователя, в какой кодировке посылать ему письма, не получится заменить две радиокнопки Windows 1251 и KOI-8 единым чекбоксом KOI-8. Пользователь не обязан понимать, в какой кодировке система будет посылать ему письма по умолчанию. К счастью, такие ситуации редки.

И чекбоксы и радиокнопки 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 радиокнопки 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.

Взаимодействие. Это может показаться невероятным, но до сих пор в интернете 99% чекбоксов и радиокнопок реализованы неправильно. Дело в том, что создатели языка HTML, ничего не понимавшие в проектировании интерфейсов, были поначалу искренно уверены в том, что в этих элементах управления нажимается только визуальный индикатор переключения, т. е. кружок или квадратик. На самом деле это совершенно не так! Нажимабельной должна быть ещё и подпись, просто потому, что закон Фитса (см. «Быстрый или точный» на стр. 10) однозначно требует больших кнопок. Но в интернете всего этого нет, поскольку в HTML конструкция чекбоксов и радиокнопок просто не позволяла делать нажимабельными подписи. Сейчас это стало технически возможным (через тег Label), но по инерции и вполне понятной лени никто чекбоксы нормальными не делает. Увы.

В интернете первым признаком профессионально разработанного интерфейса являются нажимабельные подписи к чекбоксам и радиокнопкам

Другой аспект: при необходимости заблокировать элемент, желательно визуально ослаблять не только квадрат или круг, но и подпись.

Вариант для панелей инструментов

Как чекбоксы, так и радиокнопки, бывают двух видов: описанные выше стандартные, и предназначенные для размещения на панелях инструментов (см. «Панели инструментов» на стр. 89).

Рис. 35. Пример чекбоксов и радиокнопок на панели инструментов. Слева 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 (насколько я знаю, ни в одной ОС метода визуального различения не выработано). Это не очень критично, поскольку панелями инструментов пользуются в основном сравнительно опытные пользователи, так что страдать по этому поводу не стоит. Тем не менее, на панелях инструментов полезно располагать группы радиокнопок отдельно от групп чекбоксов (чтобы они не смешивались в сознании пользователей).

Вообще говоря, графические версии чекбоксов и радиокнопок можно располагать и в диалоговых окнах. Делать это, однако, не рекомендуется, поскольку в окнах они слишком уж похожи на командные кнопки, кроме того, такие кнопки не подразумевают подписей (которые в диалоговых окнах ничего не стоят, принося в то же время явную пользу).

Обратите внимание, что на панелях инструментов чекбоксы и радиокнопки могут быть кнопками прямого действия.

Списки

Все часто используемые списки функционально являются вариантами чекбоксов и радиокнопок. Скорость доступа к отдельным элементам и наглядность в них принесены в жертву компактности (они экономят экранное пространство, что актуально, если количество элементов велико) и расширяемости (простота загрузки в списки динамически изменяемых элементов делает их очень удобными при разработке интерфейса, поскольку это позволяет не показывать пользователю заведомо неработающие элементы).

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. В таких случаях не стоит добавлять к списку горизонтальную полосу прокрутки, лучше урезать текст элементов. Для этого нужно определить самые важные фрагменты текста (например, для URL это начало и конец строки), после чего все остальное заменить отточием (...).

Поскольку нужно максимально ускорить работу пользователей, необходимо сортировать элементы. Идеальным вариантом является сортировка по типу элементов. Если же элементы однотипны, их необходимо сортировать по алфавиту, причем списки с большим количеством элементов полезно снабжать дополнительными элементами управления, влияющими на сортировку или способ фильтрации элементов. Если можно определить наиболее популярные значения, их можно сразу расположить в начале списка, но при этом придется вставлять в список разделитель, а в систему – обработчик этого разделителя.

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 сканируются быстрее «голого» текста.

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. Справа на иллюстрации крутилка (см. стр. 74).

Раскрывающийся список, как правило, вызывает две проблемы, одна появляется преимущественно в ПО, другая – в интернете.

Первая проблема заключается в том, что иногда отсутствие места на экране не позволяет использовать ни чекбоксы с радиокнопками, ни пролистываемые списки множественного выбора. Приходится делать раскрывающийся список, в котором помимо собственно элементов есть «мета-элемент», включающий все элементы из списка. Этому элементу часто не дают названия, оставляя строку списка пустой, что неправильно, поскольку требует от пользователя слишком глубокого абстрагирования. Такой мета-элемент нужно снабжать названием, например, Все значения или Ничего.

В интернете проблема иная. Раскрывающийся список часто используется как навигационное меню. Это изначально неправильно, поскольку содержимое такого меню не видно сразу и уж тем более им трудно индицировать пользователям, в каком разделе сайта они находятся. Это, впрочем, не главное. Большая проблема заключается в том, что список снабжают скриптом, который запускается сразу по выбору значения. Такой метод имеет два недостатка. Во-первых, список исторически не является элементом управления прямого действия (как и чекбокс, например), что приводит к потере пользователями чувства контроля над системой. Во-вторых, раскрывающиеся списки довольно сложны, так что пользователи часто совершают моторные ошибки при выборе нужного элемента. Поскольку эта ошибка не может быть обнаружена системой и не всегда обнаруживается пользователями, часты ситуации, когда пользователь (как ему кажется) выбирает один раздел, а перемещается в другой, что совсем нехорошо. Таким образом, навигационные раскрывающиеся списки нужно снабжать кнопкой, которая и будет запускать действие, запускать же действие сразу после выбора элемента в списке нельзя.

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.

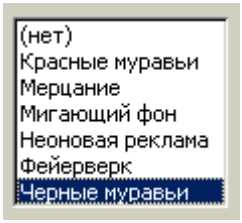


Рис. 37. Список единственного выбора. Обратите внимание, что в ситуациях, когда все элементы помещаются в список без пролистывания, список работает в точности как группа радиокнопок.

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. Проблема в том, что такой список в браузере будет выглядеть как список единственного выбора, более того, чтобы выбрать несколько элементов пользователю придется удерживать клавишу Ctrl. Это значит, что воспользоваться таким списком сможет только малая часть аудитории (и даже наличие подсказки у списка положения не исправит). Из-за такой убогой реализации списков браузерами, использовать их, как правило, оказывается невозможно. Приходится использовать чекбоксы[1].

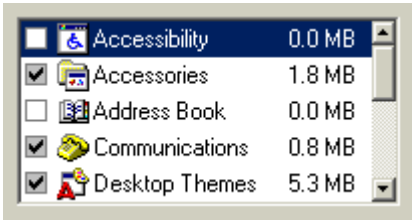


Рис. 38. Список множественного выбора с чекбоксами.

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.

1. В принципе, с появлением слоев, внедренных окон (IFrame) и Macromedia Flash, появилась возможность самостоятельно создавать такие списки и в интернете. К сожалению, это довольно трудоемкое занятие, к тому же запрограммировать все возможности списка (например, перемещение по списку с клавиатуры) практически нереально.

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.

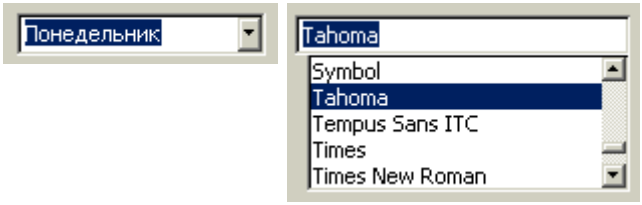


Рис. 39. Раскрывающийся комбобокс с установленным фокусом ввода (слева) и расширенный комбобокс (справа).

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. Во-вторых, что гораздо хуже, раскрывающиеся комбобоксы отсутствуют в интернете как класс. Поддержки их нет ни в браузерах, ни в HTML.

Проблемы расширенных комбобоксов, напротив, совершенно иные. Их с трудом, но можно реализовать в интернете (через JavaScript). Они имеют уникальный вид, отличающий их от остальных элементов управления. Зато их сравнительно трудно (хотя и гораздо легче, чем в интернете) реализовать в ПО. При этом расширенный комбобокс потребляет много места на экране.

Поскольку комбобоксы являются гибридами списков и полей ввода, к ним применимы те же требования, что и к их родителям.

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.

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. © РОЛ

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, и при этом объем вводимого текста ограничен, пользователи неприятно изумляются, обнаружив, что они не могут ввести текст, хотя место под него на экране имеется. Соответственно, вообще нельзя делать поле ввода шире максимального объема вводимого в них текста.

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, моё личное мнение заключается в том, что, поскольку восприятие подписей занимает определенное время, которого жаль, лучше всего 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. Если очень жалко экранное пространство, этим методом стоит пользоваться.

Крутилки

Крутилка (spinner, little arrow) есть поле ввода, не такое универсальное, как обычное, поскольку не позволяет вводить текстовые данные[1], 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, см. «Предсказание скорости» на стр. 120), к тому же ещё и сбивает фокус внимания, отсутствие нужды в клавиатуре оказывается большим благом. Во всяком случае, ввод значения в крутилку с клавиатуры достаточно редок, т. е. пользователи воспринимают крутилки целиком и полностью положительно. Во-

1. Точнее, позволяет, но при этом оказывается ничем не лучше обычного поля ввода.

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.

К сожалению, в интернете нет специального элемента для крутилки. Сделать элемент, похожий на крутилку, можно без труда, создав список множественного выбора высотой в один элемент, но ввод в него с клавиатуры будет невозможен. К счастью, крутилку можно с относительно небольшими затратами сделать в Macromedia Flash.

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 (см. рис. 42)
- нужно передать пользователям ранжируемость значений.

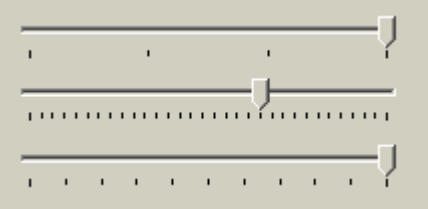


Рис. 42. Примеры ползунков. Видно, что количество параметров в ползунке может быть весьма значительным.

- 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).

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

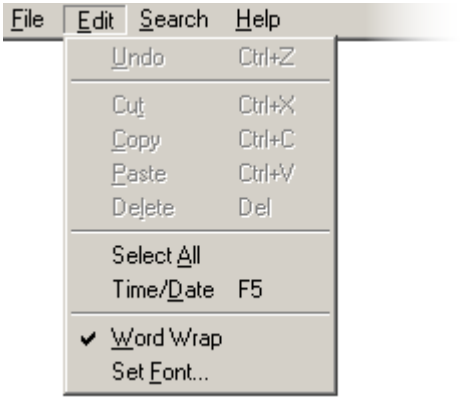


Рис. 43. Стандартное раскрывающееся меню.

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

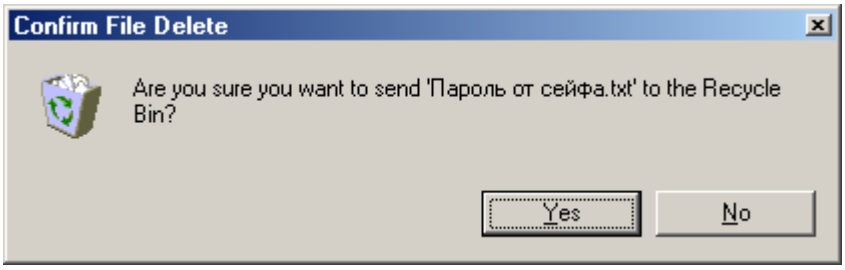


Рис. 44. Это тоже меню.

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 даже обнаружено, что меню является самым эффективным средством обучения[1]). 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. 67), 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 раскрывающееся 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 высокую скорость работы, лучше обучают пользователей, но зато занимают место на экране. 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. 98), 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.

1. Diana Parton, Keith Huffman, Patty Pridgen, Kent Norman, and Ben Shneiderman. Learning a Menu Selection Tree: Methods Compared. Behaviour and Information Technology, 4 (2) 1985, pp. 81-91.

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[1].

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» (так же, 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 КВП.

1. По этой же причине не рекомендуется что-либо делать со строкой статуса (см. стр. 88): в интернете это практически единственный способ узнать, куда приведет ссылка, не идя по ней.

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. 118).

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 меню[1].

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 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 с новыми элементами меню.

1. Напротив, с научной точки зрения оценить преимущество ширины перед глубиной (и обратно) достаточно сложно. На эту тему было написано множество работ, было проведено множество исследований, но четкого и однозначного вывода сделать из них нельзя.

Видно, что действия пользователя при поиске нужного элемента отчетливо цикличны, при этом на каждом шаге есть вероятность ошибок. 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, пожалуй, лучшее, что дал интернет науке о проектировании интерфейсов: в качестве аннотации к элементу можно показывать наиболее популярные элементы следующего уровня.

Рис. 45. Насколько я знаю, такой элемент управления для меню впервые появился в каталоге Yahoo. Несмотря на то, что с тех пор этот элемент сейчас присутствует на множестве сайтов, я, стремясь подчеркнуть заслугу Yahoo, выбрал оригинал. © Yahoo! Inc

В этом случае пользователь может сформировать контекст элемента, не перемещаясь внутрь этого элемента, при этом вероятность ошибочного перехода значительно снижается. Помимо уменьшения числа ошибок, такая система позволяет ускорить доступ к наиболее популярным элементам второго и последующих уровней.

В целом, ширина и глубина меню являются, пожалуй, наименее значимыми факторами. Гораздо важнее хорошая группировка, при этом как группировку, так и структуру дерева меню, всё равно лучше определять карточной сортировкой (см. стр. 118).

Применительно же к раскрывающимся меню действует ещё один ограничитель глубины. Раскрывающиеся меню довольно тяжелы в использовании, поскольку требуют от пользователей достаточно тонкой моторики. Поэтому главное меню с более чем тремя уровнями вложенности просто невозможно.

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 окна или экраны, окна требуют не меньше заботы, чем элементы управления.

Типы окон

Буржуазная псевдонаука знает несколько типов окон, а именно:

- главные окна программы
- окна документа
- режимные диалоговые окна (о разнице между режимными и безрежимными окнами см. стр. 85)
- 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 режимные диалоговые окна (которые разработчики поленятся переделывать). Но об этом отдельно.

Недолгая история окон на экране

Сейчас многим в это трудно поверить, но сравнительно недавно никаких окон не было, даже диалоговых окон, которые уже стали восприниматься как данность. Вместо них какая-то часть экрана выделялась под меню[1], которое в те времена было функционально более богатым, чем меню теперешнее (так, нормой были поля ввода в меню).

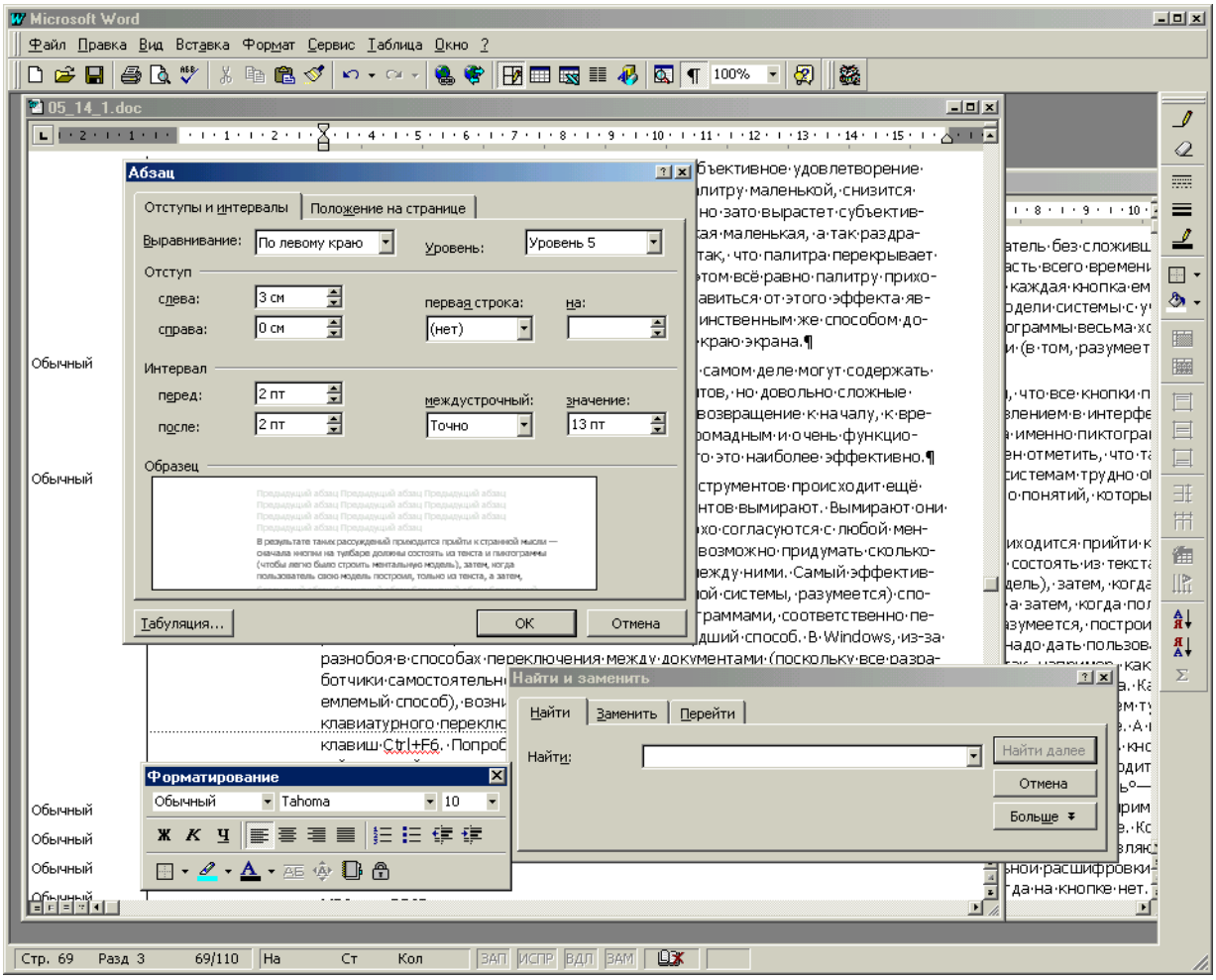


Рис. 46. Типы окон на примере MS Word 97. Самое большое окно есть окно программы. Внутри него два окна документов, в более свежих версиях Word их уже нет. Слева сверху располагается режимное диалоговое окно (Абзац), под ним справа – безрежимное (Найти и заменить). Слева внизу располагается палитра (Форматирование, я получил её, оторвав соответствующую панель инструментов от края окна). Сверху и справа две панели инструментов (бывшие палитры).

Потом, с появлением графического режима, стало возможным реализовать в интерфейсе метафору рабочего стола (если бумажные документы могут лежать на столе друг на друге, то почему этого не могут делать электронные документы?). Появились окна программ, окна документов и диалоговые окна, первоначально сплошь режимные.

Понятие «режимное диалоговое окно» кажется довольно загадочным (еще более загадочным кажется его англизированный вариант «модальное диалоговое окно», из-за чего я этот вариант не употребляю). На самом деле всё просто. Если открывшееся окно блокирует доступ к остальной части системы, происходит, фактически, запуск нового *режима работы* (поскольку

1. Формально, называть словом «меню» то, что было, не совсем правильно (собственно говоря, меню в нашем понимании там отсутствовало как класс). Но поскольку располагалось это «нечто» на месте современных меню, я и выбрал такое название.

функциональность отдельного диалогового окна никогда не совпадает с функциональностью системы в целом). После того, как окно закрыто, происходит возвращение предыдущего (основного) режима. В этом и есть всё значение термина «режимный».

Прошло несколько лет, и наличие режима в диалоговых окнах стало немодным. Во-первых, всех раздражает, что, вызвав диалоговое окно и обнаружив, что вызвано оно преждевременно, приходится закрывать окно и открывать его в следующий раз заново. Во-вторых, что важнее, в системах, ориентированных на документы, режим сбивает внимание пользователя и вообще лишает его ощущения управляемости (в отличии систем, ориентированных на формы ввода, в которых режим работает лучше, чем его отсутствие). В-третьих, сама по себе идея сближения интерфейса с реальным миром (в частности, метафора рабочего стола) протестовала против идеи режимов в любом их проявлении, поскольку в реальном мире вообще не бывает режимов, аналогичным интерфейсным. А поскольку «дизайн пользователей» был ориентирован на функционирование в реальном мире, решили не переделывать пользователей, а переделать интерфейс.

Избегайте режимов работы

Так появились безрежимные диалоговые окна, т. е. окна, которые можно было неограниченное время держать на экране, переключаясь по мере надобности между ними и собственно документом. К сожалению, и здесь не без проблем. Дело в том, что такие диалоговые окна нельзя делать тонущими, т. е. позволять пользователю перекрывать их окнами документа или программы. Причина проста – пользователи забывают, что они когда-то открывали соответствующее окно и пытаются открыть его заново. Зачем, спрашивается, такие окна? Поэтому решили сделать такие окна плавающими, т. е. перекрываемые только другими плавающими окнами этой же программы или другими программами. Разумеется, некоторые диалоговые окна невозможно сделать безрежимными: например, что делать с сообщениями об ошибках? Но, в целом, с переводом окна в безрежимное состояние нет особой проблемы.

Но и тут обнаружилась гадость. Дело в том, что просто диалоговое окно, даже будучи безрежимным, малополезно, поскольку перекрывает слишком много важного и нужного. Решение этой проблемы было эволюционным, а не революционным, и поэтому относительно простым – были придуманы палитры, т. е. окна, из которых выжали всё пустое место. Сразу оказалось, что палитры, помимо малых размеров, имеют одно большое достоинство: пользователи очень любят их расставлять на экране индивидуальным порядком. Пользы это особой не приносит, зато существенно повышает субъективное ощущение контроля над системой. К сожалению, визуальный дизайн палитр, как правило, довольно сложен и длителен, так что сугубо экономические причины мешают переделать в палитры все диалоговые окна.

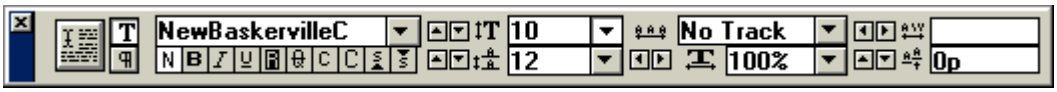


Рис. 47. Пример палитры из программы Adobe PageMaker. К сожалению, из-за малых размеров палитр в них никогда не помещаются полноценные подписи к элементам, что существенно замедляет скорость обучения. Например, несмотря на то, что я пользуюсь PageMaker уже много лет, я до сих пор не знаю, что делает большая квадратная кнопка слева. © Adobe

Как легко догадаться, гадость была найдена и в палитрах. Существует неформальный, но на удивление верный закон[1], гласящий, что субъективная важность информации, перекрываемой диалоговым окном (палитрой в

частности), не зависит ни от размеров, ни от положения окна, а зависит только от периметра. В результате постоянно оказывается, что пользователи, стараясь открыть нужную информацию, перекладывают окна с места на место, что снижает производительность (несущественно) и субъективное удовлетворение (существенно). При этом, если сделать палитру маленькой, снизится вероятность её вынужденного перетаскивания, но зато вырастет субъективное неудовольствие от её перетаскивания («такая маленькая, а так раздражает»). Более того. Гораздо чаще оказывается так, что палитра перекрывает не всю нужную информацию, но её часть; при этом всё равно палитру приходится перемещать. Единственным способом избавиться от этого эффекта является уменьшение периметра палитры, а добиться этого можно, только прикрепив палитры к краю экрана.

Так родились панели инструментов, которые на самом деле могут содержать (и содержат) не только пиктограммы инструментов, но довольно сложные элементы управления. В некотором смысле, это возвращение к началу, к временам, когда один из краев экрана был занят громадным и очень функциональным меню. С другой стороны, оказалось, что это наиболее эффективно.

Параллельно с рождением сложных панелей инструментов происходит ещё одна драма борьбы за выживание. Окна документов вымирают[1]. Вымирают они по двум простым причинам. Во-первых, они плохо согласуются с ментальной моделью большинства пользователей. Во-вторых, невозможно придумать сколько-нибудь эффективного способа переключаться между ними. Самый эффективный (с точки зрения разработчиков операционной системы, разумеется) способ обычно отдается переключению между программами, соответственно, переключению документов достается заведомо худший способ. В Windows, из-за разнобоя в способах переключения между документами (поскольку все разработчики самостоятельно старались найти какой-либо приемлемый или неприемлемый способ), возникают трогательные казусы: в MS Word, например, официальным способом для клавиатурного переключения между документами является комбинация клавиш Ctrl+F6. Попробуйте использовать эту комбинацию клавиш одной рукой, и вы поймете, что это невозможно. Главное же в другом: с самого начала окна документов были не столько желаемым свойством системы, сколько хаком. Для того чтобы запустить две одинаковые программы, каждая с одним документом внутри, не хватало ресурсов компьютера, вот и приходилось запускать одну программу с двумя документами. Сейчас, напротив, памяти достаточно, к тому же появились технологии программирования, позволяющие ни о чем таком даже не думать. Так что окна документов умирают. Не будем им мешать.

1. Его открыл я.
1. Во всех операционных системах, кроме Mac OS, устройство которого таково, что разницы между окном программы и окном документа не было никогда.

Окна, помимо областей с элементами управления, имеют некоторые общие элементы, главными из которых являются строки заголовка окна, строки статуса, панели инструментов и полосы прокрутки.

Строка заголовка окна

У каждого окна есть строка заголовка. Человеку не свойственно обращать внимание на обыденность, особенно если эта обыденность не находится в фокусе его внимания (а строка заголовка как раз в нем не находится). Поэтому пользователи строкой заголовка интересуются весьма мало. В результате, пользователи обращают внимание на строку заголовка, только обучаясь пользоваться компьютером или в ситуациях, когда они совсем ничего не понимают в системе. Из этого, однако, не следует, что строкой состояния можно пренебрегать. Точнее, самой строкой как раз пренебречь можно, но её содержимым – нельзя.

Дело в том, что текст и, в меньшей степени, пиктограмма заголовка играют важную роль в ПО (они заведуют переключением задач) и очень важную в интернете (заведуют навигацией).

Рис. 48. Влияние строки заголовка окна на переключение между задачами. Панель задач в Windows создает по кнопке для каждой запущенной программы. Поскольку ширина экрана ограничена, при увеличении количества запущенных программ размеры кнопок сокращаются, соответственно в эти кнопки помещается меньше текста. В результате пользователь сохраняет способность опознать программу по её пиктограмме и обрывку текста, но теряет возможность опознавать документы (обратите внимание, что даже в верхнем примере полное название документа определить невозможно). Проблемы можно было бы избежать, если бы название программы на кнопке (и в строке заголовка окна) было бы короче и/или название документа выводилось бы *до* названия программы. Заодно пользователям не пришлось бы сотни и тысячи раз читать название программы (последствия неумеренного продвижения торговой марки).

С переключением задач всё просто и сложно одновременно. Просто, поскольку правило тут простое «Релевантное выводится в первую очередь». Поскольку пользователю нужен именно конкретный документ конкретной программы, а вовсе не программа просто (мы уже определили, что окна документов, не попадающие в переключатель задач, нехороши), названия документов, как более релевантные, нужно выводить в первую очередь. Наоборот, сложность состоит в том, что из-за жесткости интерфейса Windows много не сделаешь. Тем не менее, сокращать название программы нужно безусловно.

Иная ситуация в интернете. Поскольку пиктограмма в строке заголовка приходит от браузера, нет особой возможности оптимизировать переключение задач. С другой стороны, качество этого заголовка оказывает существенное влияние на навигацию, поскольку при показе результатов поиска в поисковых системах заголовком элемента становится содержимое тега Title. Каковое содержимое и попадает в обычном режиме на верх экрана. При этом в интернете нет проблемы с текстом заголовка – что хотим, то и пишем (стараясь не обращать внимания на то, что к этому прибавится название браузера).

Нажатие на пиктограмму в строке заголовка вызывает раскрывающееся меню, являющееся замечательным местом для вызова функций, которые нужны только наиболее опытной аудитории

Правило релевантности действует и здесь – в начале строки должна быть более релевантная информация, нежели в её конце. Поскольку связки «программа-документ» в интернете нет, эффективнее всего показывать адрес текущей страницы в навигационной системе сайта (если сайт иерархический). В данном случае релевантность требует, чтобы сначала шло название текущего документа, затем раздела, в котором он находится, затем раздела более высокого уровня и так далее. Не надо также забывать, что размер строки ограничен, так что более 70-80 символов в ней быть не может.

Также важно понимать, что тот факт, что пользователи редко читают заголовки окна, вовсе не означает, что заголовки пользователям не нужны. Напротив, хороший заголовок может здорово облегчить понимание работы диалога. Поэтому наличие на экране заметного и адекватного заголовка окна часто оказывается очень полезным. Жалко только, что в обычном Windows-интерфейсе места под него нет.

Строка статуса

Строка статуса является, пожалуй, самым недооцененным элементом интерфейса (во всяком случае, способы её использования в интернете существенно портят статистику). В то же время она заслуживает лучшей участи.

Почему-то распространено мнение, будто строка статуса предназначена для того, чтобы информировать пользователей о значении тех или иных элементов интерфейса. Подразумевается, что если пользователь подводит курсор к какому-либо элементу, в строке статуса появляется краткое его описание. На самом деле строка не может этого делать вообще: дело в том, что курсор находится в одном месте, а подсказка появляется совсем в другом, пользователю при этом приходится читать подсказку либо переводя взгляд, либо периферийным зрением. Разумеется, никто в таких условиях читать подсказку не будет, причем те, кто уверен, что строка статуса есть место для подсказки, чувствуют это прекрасно. Неудивительно, что разработчики строку статуса игнорируют.

В действительности строка статуса предназначена для двух вещей: она может быть либо собственно строкой статуса, т. е. отображать текущее состояние системы, либо быть панелью инструментов для опытных пользователей (или же делать и то, и другое). Разберем это подробнее.

Отображение текущего состояния системы. Практически каждая система имеет свойства, либо зависящие от документа, либо изменяющиеся со временем. Например, в иллюстративных программах объекты имеют какие-либо свойства, причем не все эти свойства показываются. Другой пример: когда система долгое время занята, она должна показывать пользователю индикатор степени выполнения. И, наконец, самый простой пример: пользователь текстового процессора имеет право знать, на какой странице документа он сейчас находится. Эффективнее всего выводить всё это в строке статуса.

Рис. 49. Статусная строка Adobe PhotoShop. Слева отображается текущий масштаб отображения документа, вслед за ним объем занимаемой документом памяти (стрелка переключает тип показываемой информации), затем индикатор степени выполнения, а справа – контекстная подсказка (место оставалось, вот его и заполнили).

Строка статуса особенно интересна как место вывода индикатора степени выполнения. Существует занятная закономерность: по месту вывода индикатора выполнения можно определить качество интерфейса системы: если индикатор выводится в строке статуса, то система обладает в целом хорошим интерфейсом, если же индикатор выводится в другом месте – не столь уж хорошим.

Панель инструментов для опытных пользователей. Зачастую система обладает функциональностью, которая с одной стороны важна, а с другой – способна свести с ума неподготовленного пользователя. Обычно это касается не столько собственно функций, сколько режимов работы системы, о недостатках которых уже говорилось.

Так, любой *не очень опытный* пользователь MS Word, не имея под рукой эникейщика, потратит как минимум минут пятнадцать на выход из режима замены текста, предварительно нечаянно этот режим включив. Правильнее всего, конечно, включать такие режимы из меню, рассчитывая, что пользователь вряд ли выберет случайно элемент третьего уровня, но если в этот режим надо входить часто, меню спасать перестает.

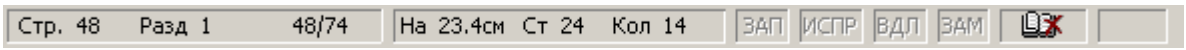


Рис. 50. Строка статуса MS Word. Первые два блока показывают положение открытого фрагмента документа, за ними идут четыре переключателя режимов, затем кнопка, нажатие на которую пролистывает документ до ближайшей языковой ошибки (это ещё и индикатор системы проверки правописания).

В таких случаях строка состояния является отличным решением проблемы. С одной стороны, делая переключатели режимов непохожими на поля вывода (знаете ли вы, например, что метки ЗАП, ИСПР, ВДЛ и ЗАМ в статусной строке MS Word не только индикаторы?) можно снизить вероятность ошибочного переключения. С другой стороны, если уж пользователь нечаянно щелкнет на переключателе, он сразу же увидит изменение его вида и впоследствии, вероятно, сможет переключиться назад. С еще одной стороны, опытный пользователь сможет переключаться между режимами так же легко, как если бы он переключался через панель инструментов.

Панели инструментов

Все панели имеют следующие достоинства:

- они позволяют пользователям быстро вызывать нужные функции мышью
- они позволяют пользователям меньше задействовать память
- они повышают визуальное богатство интерфейса
- они ускоряют обучение работе с системой (по сравнению с раскрывающимся меню) благодаря своей большей наглядности.

Зато они имеют и недостаток: занимают много места на экране, так что поместить в них всё, что хочется, невозможно. Решить эту проблему можно двояко. Во-первых, можно (и нужно) помещать в панель только наиболее часто используемые команды (поддерживая это решение возможностью индивидуальной настройки панели пользователем). Во-вторых, панель можно сделать зависимой от контекста действий пользователя. Оба способа не противоречат друг другу, так что использовать стоит оба.

Панель инструментов нежелательно делать единственным способом вызова функции

В настоящее время нет технической проблемы с помещением в панели произвольных элементов управления (остался только один ограничитель – размер помещаемых элементов), так что последние преграды, мешавшие делать сложные панели, исчезли. Этим стоит пользоваться, поскольку это позволяет экономить время, уходящее на открытие и закрытие диалоговых окон, и повышать интегральное качество взаимодействия с системой (пользователям *нравится* пользоваться сложными панелями).

Текст на кнопках. Самыми частыми элементами управления, размещаемыми на панелях инструментов, является командные кнопки, при этом их использование отличается от обычного. Дело в том, что места настолько не хватает, что очень хочется заменить текст кнопок пиктограммами. Но это не так просто.

Дело в том, что когда приходит время совершить выбор, имея в качестве альтернатив визуальные объекты, «человек выбирающий» чаще всего транслирует эти объекты в звуки, а именно в слова (в голове, разумеется). Затем эти слова помещаются в кратковременную память, в дело включается собственно сознание (предыдущие этапы проходят на бессознательном уровне) и выбирает нужный объект. Применительно к реальной жизни это значит, что пользователь, глядя на панель с пиктограммами, видит скорее не пиктограммы, но слова. Но не всегда.

- **Случай 1.** Опытный пользователь, уже знающий, где на панели находится нужная кнопка, знающий её значение, при этом выбор действия уже произведён при помощи сложившейся ментальной модели. В такой ситуации слова пользователю уже не важны, важно отличие нужной ему кнопки от остальных. Т. е. такому пользователю даже уже все равно, что на пиктограмме изображено, лишь бы она выглядела максимально контрастно (чтобы ускорить её поиск).
- **Случай 2.** Опытный пользователь, обладающий сложившейся ментальной моделью, но не знающий, где конкретно расположена нужная ему кнопка и как она выглядит. Выбор действия уже произведен, осталось только найти нужную кнопку. При этом пиктограмма оказывается ненужной, так как в качестве матрицы пользователь использовать её не может (поскольку не знает, как она выглядит). Более того, поскольку пользователь ищет слово из содержимого своей кратковременной памяти, каждая пиктограмма будет его без пользы отвлекать, при этом пользователь будет тратить время на расшифровку смысла всех попадающихся ему на пути пиктограмм.
- **Случай 3.** Неопытный пользователь без сложившейся ментальной модели. Такой пользователь большую часть всего времени тратит на поиск нужной ему кнопки, а также, поскольку каждая кнопка ему внове, на постоянное улучшение своей ментальной модели системы с учетом своих новых открытий. В таких случаях пиктограммы лучше текста, но не заменяют его, так как помогают *быстрее* понять действие кнопки (в том, разумеется, случае, когда пиктограмма адекватна смыслу действия).

В результате таких рассуждений приходится прийти к странной мысли – сначала кнопки на панели инструментов должны состоять из текста и пиктограммы (чтобы легко было строить ментальную модель), затем, когда пользователь свою модель построил, только из текста, а затем, когда пользователь окончательно обучился пользоваться системой, только из пиктограммы. Разумеется, построить такую систему невозможно, так что приходится определяться. Поскольку в двух случаях из трех текст оказывается нужен (тем более что начинающие и средне продвинутые пользователи составляют большинство), удалять его из панели оказывается неправомерным.

Здесь действует ещё и вездесущий закон Фитса. Поскольку кнопка с пиктограммой и текстом всегда больше кнопки с текстом или пиктограммой просто, она оказывается более эффективной в отношении скорости, поскольку в неё легче попасть мышью.

Таким образом, эффективнее всего (учитывая все аргументы за и против) делать кнопки на панелях инструментов диалектически: самые главные кнопки нужно делать парой «пиктограмма плюс текст», а остальные в зависимости от их направленности – функции для опытных пользователей пиктограммами, а для неопытных текстом.

Когда графических интерфейсов еще не было, пользователи перемещались по документу с помощью клавиатуры. С тех далёких времен на клавиатуре остались клавиши Home и End, равно как Page Up и Page Down. В целом, пользователи были удовлетворены своей судьбой (благо, они не знали альтернатив). Затем появились графические интерфейсы. Первым делом были придуманы полосы прокрутки. К сожалению, оказалось, что они работают не слишком хорошо.

Проблема полос прокрутки заключается в следующем: для маленьких документов они не очень нужны, поскольку пользователям, держащим руки на клавиатуре, гораздо легче переместиться к нужному фрагменту с помощью клавиш со стрелками. Напротив, в больших документах малое перемещение ползунка приводит к существенному сдвигу области просмотра, так что после перемещения нужно еще и подправляться либо клавиатурой, либо стрелками на полосе прокрутки. Более того: во многих случаях невозможно реализовать динамическое изменение области просмотра во время перемещения ползунка, а значит, перемещаться по большим документам приходится в несколько шагов. И еще раз более того: предположим, что это динамическое изменение всё-таки есть. Тогда пользователю нужно: сначала перевести взгляд на ползунок, затем курсор на ползунок, затем взгляд на документ и только потом, перемещая мышь вверх или вниз, следить за областью просмотра, на тему «не пора ли отпустить курсор».

Пользователи ненавидят горизонтальные полосы прокрутки

Нечего и говорить, что пользователи избегают пользоваться полосками прокрутки (тем более что курсорные клавиши никто с клавиатуры не убирал). Фактически, чуть ли не единственным применением этих полосок является перемещение «примерно к нужному фрагменту» при работе с большими документами.

Разумеется, такое положение вещей никого особенно не радовало. Поэтому было придумана «дополнительная стоимость» полосок – размер ползунка был сделан пропорциональным отношению видимой части документа ко всему его объёму. Это очень хорошая и полезная функция, поскольку она позволяет использовать полосы прокрутки не только как элемент управления, но и как индикатор размера документа, благодаря чему степень бесполезности полосок значительно снижается. Помимо этого было придумано довольно много других дополнительных стоимостей, так, например, на полоске прокрутки можно отображать границы разделов документа.

Полосы прокрутки без индикации размера документа практически бесполезны

Тем не менее, всё это так и не сделало полосы прокрутки здорово лучше: как и раньше, полосы не столько помогают перемещаться по документу, сколько показывают то, что не весь документ помещается на экране. Решение этой проблемы пришло с несколько непривычной стороны, во всяком случае, графический пользовательский интерфейс не пригодился – была придумана мышь с колесиком прокрутки. Решение это чуть ли не идеальное, поскольку не требует от пользователя переносить внимание с документа на элемент управления. Конечно, для перемещения по большим документам колесо не слишком эффективно (палец устаёт), но малые и средние перемещения получаются замечательно, тем более что процент больших документов невелик. Поскольку мышь стоит не слишком дорого, а служит не слишком долго, сейчас можно смело рассчитывать на наличие у пользователей мышей с колесиком (я знаю двух людей, которые пошли в магазин и купили себе новые мыши сразу после того, как увидели мышь с колесом у сослуживца).

Таким образом, полосы прокрутки стали совсем уж бесполезны, поэтому относиться к ним надо не как к данности, но как к еще одному элементу управления, который можно использовать, а можно и не использовать. При этом есть как аргументы в пользу использования, так и существенный аргумент против него – полоски прокрутки занимают много места на экране. Ладно еще, когда на экране одна полоска, а что делать, если их три или более?

С появлением мышей с колёсиками, полоски прокрутки смело можно делать тоньше

К сожалению, вовсе не использовать полосы прокрутки в ПО затруднительно, MS Windows User Experience прямо заставляет разработчика ими пользоваться. В интернете ситуация иная – никто никого не заставляет. Осталось разобраться, как же сделать пролистывание документа идеальным.

Если всё-таки приходится оставлять полосы прокрутки, крайне желательно добиться нескольких свойств полосок:

- Размер ползунка должен показывать общий объем пролистываемого документа.
- Стрелки на полосах должны быть спаренными, т. е. обе стрелки должны находиться рядом, а не на разных сторонах полоски. Это один из случаев, когда логичность интерфейса вступает в противоречие с эффективностью. Если при перелистывании была допущена ошибка, спаренные кнопки позволяют минимизировать перемещение курсора к стрелке, ведущей в обратную сторону (см. рис. 51).
- Если невозможно сделать динамическое изменение области просмотра при пролистывании, необходимо показывать текущее местоположение пользователя во всплывающей подсказке (см. рис. 51). Обратите внимание, что местоположение подсказки при перемещении курсора должно оставаться неизменным.
- Необходимо обеспечить обработку погрешности перемещения курсора. Когда пользователь курсором перемещает ползунок, а смотрит в это время на документ, курсор может сойти с полосы. До определённого момента (смещение на 30-70 пикселей) система должна такое смещение игнорировать.

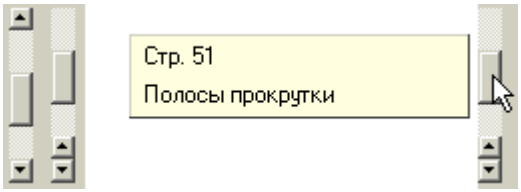


Рис. 51. Два способа улучшить полоску прокрутки. Слева стандартная полоска, рядом с ней полоска со спаренными стрелками. Справа: всплывающее сообщение, показывающее, какому фрагменту документа соответствует текущее положение ползунка.

Теперь об альтернативных элементах управления. Чаще всего используются кнопки со стрелками, т. е. фактически полоски прокрутки, из которых вырезано самое главное. Это не очень хороший элемент, потому что он совершенно линеен: когда пользователь нажимает на кнопку со стрелкой, документ листается с фиксированной скоростью, изменить которую пользователь не в силах. Это приводит либо к медленному пролистыванию, либо к низкой точности. Поэтому гораздо эффективнее малюсенький джойстик, часто встречающийся в ноутбуках, который в народе ласково называется клитором.

Сущность этого элемента проста: на экране располагается объект, нажатие на который меняет курсор на изображение направленных в разные стороны стрелок. Если пользователь перемещает курсор с нажатой

кнопкой мыши в сторону, документ в эту же сторону и прокручивается, причем скорость прокрутки пропорциональна расстоянию, на которое перемещен курсор. Важно только не забывать его блокировать, когда пролистывать нечего. Такой элемент управления в настоящее время реализован в MS Windows и доступен по нажатию средней кнопки мыши.

Структура окна

Структура и само устройство окна или экрана является, пожалуй, самым существенным фактором, влияющим на качество интерфейса в этом окне. Например, производительность пользователей порой можно повысить вдвое, просто изменив расположение элементов управления, не меняя сами эти элементы.

Большинство руководств по проектированию интерфейсов, перечисляя требования к структуре окна, ограничиваются замечанием, что терминационные кнопки (т. е. командные кнопки, управляющие окном, например Ok или Закрыть) должны быть либо снизу окна, либо в правой его части. Это хорошо, но мало. На самом деле всё сложнее.

Во-первых, окно должно хорошо сканироваться взглядом, т. е. его основные части должны быть сразу видны и заметны. Как правило, в окнах с малым количеством элементов управления проблем со сканированием не возникает. Проблемы появляются в больших окнах, дающих доступ ко многим функциям. Понятно, что сваливать эти функции в кучу неэффективно, для этого интерфейсные элементы должны быть организованы в блоки. В ПО для этого используются в основном рамки группировки, в интернете – пустоты, разграничивающие отдельные функции. При этом рамки удобнее в производстве, но, поскольку они являются визуальным шумом, однозначно рекомендовать их нельзя. В целом, разграничивать блоки пустотами предпочтительней (но и сложней).

Во-вторых, окно должно *читаться*, как текст. При прочих равных, окно, все элементы управления которого можно без труда связно прочесть, будет лучше запомнено и быстрее обработано мозгом (поскольку не придется преобразовывать грамматику окна в грамматику языка).

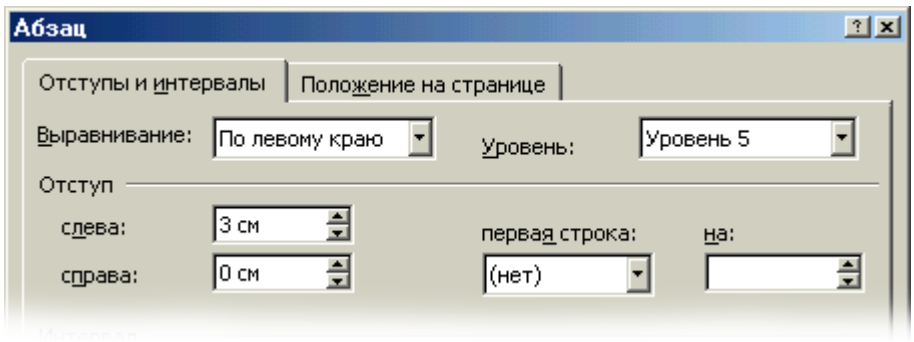


Рис. 52. Пример читаемого окна. Читается он следующим образом: «Текст выравнивается по левому краю, уровень пятый, отступ слева 3 см, справа 0 см, первая строка нет, на 5 и так далее». На этом примере прекрасно видны все неопределенности в окне: например, не говоря уже о том, что непонятно, чего именно пятый уровень, видно, что подписи к «первая строка» и к «на», расположенные сверху, разрывают единый по смыслу элемент управления на два разных.

При этом один элемент управления должен однозначно преобразовываться в единый фрагмент предложения, а единая группа элементов – в целое предложение.

Окно должно читаться, как текст

Проверить читаемость окна исключительно просто: окно нужно просто прочитать. При этом становится понятно, какие нужны подписи к элементам, как они должны быть расположены и тому подобное.

В-третьих, оно должно удовлетворять великому закону «релевантное – вперед». Чаще всего используемые элементы должны быть расположены в левой верхней части экрана, реже используемые – в правой нижней части. Обратите внимание, что окно сканируется взглядом точно так же, как происходит процесс чтения – сначала взгляд переходит в левый верхний угол, затем перемещается вправо, затем переходит на следующую «строку» и т.д. Поэтому, например, вертикальный элемент управления, разрывающий эти воображаемые строки на части, будет всегда замедлять сканирование окна (и вызывать неудовольствие у пользователей).

Не забывайте проверять диалоговые окна на нормальную работу в режиме Large Font

Теперь, возвращаясь к началу, пора объяснить, почему терминационные кнопки должны быть расположены внизу или справа, тем более что здесь действует всеобъемлющий закон. Дело в том, что в интерфейсе всегда должно быть реализовано правило: сначала выбор параметров, а затем действие (интересно, что в большинстве языков ситуация обратная, например, мы не говорим «Петю укусить», но говорим «укусить Петю»). Нарушение этого правила существенно повышает количество человеческих ошибок и ослабляет пользовательское ощущение контроля (что грозит низким субъективным удовлетворением). Это правило, будучи применено к диалоговым окнам, и заставляет помещать терминационные кнопки снизу или справа, т. е. в области, которая сканируется последней.

Увеличиваем площадь

Площадь экрана ограничена, напротив, количество элементов управления, которых может понадобиться уместить в едином функциональном блоке (т. е. окне), не ограничено ничем. В этом случае приходится использовать вкладки (см. рис. 52). Чтобы правильно их использовать, нужно соблюдать определенные требования.

Первая вкладка и остальные вкладки. Помимо умещения максимального количества элементов управления в диалоговом окне, вкладки могут выполнять еще одну роль, а именно скрывать от неопытных пользователей не очень нужную им функциональность. Проблема заключается в том, что когда нужно просто уместить в окно побольше элементов, вкладки скрывают от пользователей функциональность, возможно, что и нужную.

Некогда в Windows было два способа поместить в диалоговое окно больше, чем в него могло влезть. Можно было воспользоваться вкладками, а можно было нажать на специальную кнопку, которая увеличивала размер окна и открывала доселе скрытые элементы управления. Microsoft эти кнопки разонравились и, начиная с Windows 95, она планомерно удалила их из почти всех своих продуктов, заменив вкладками.

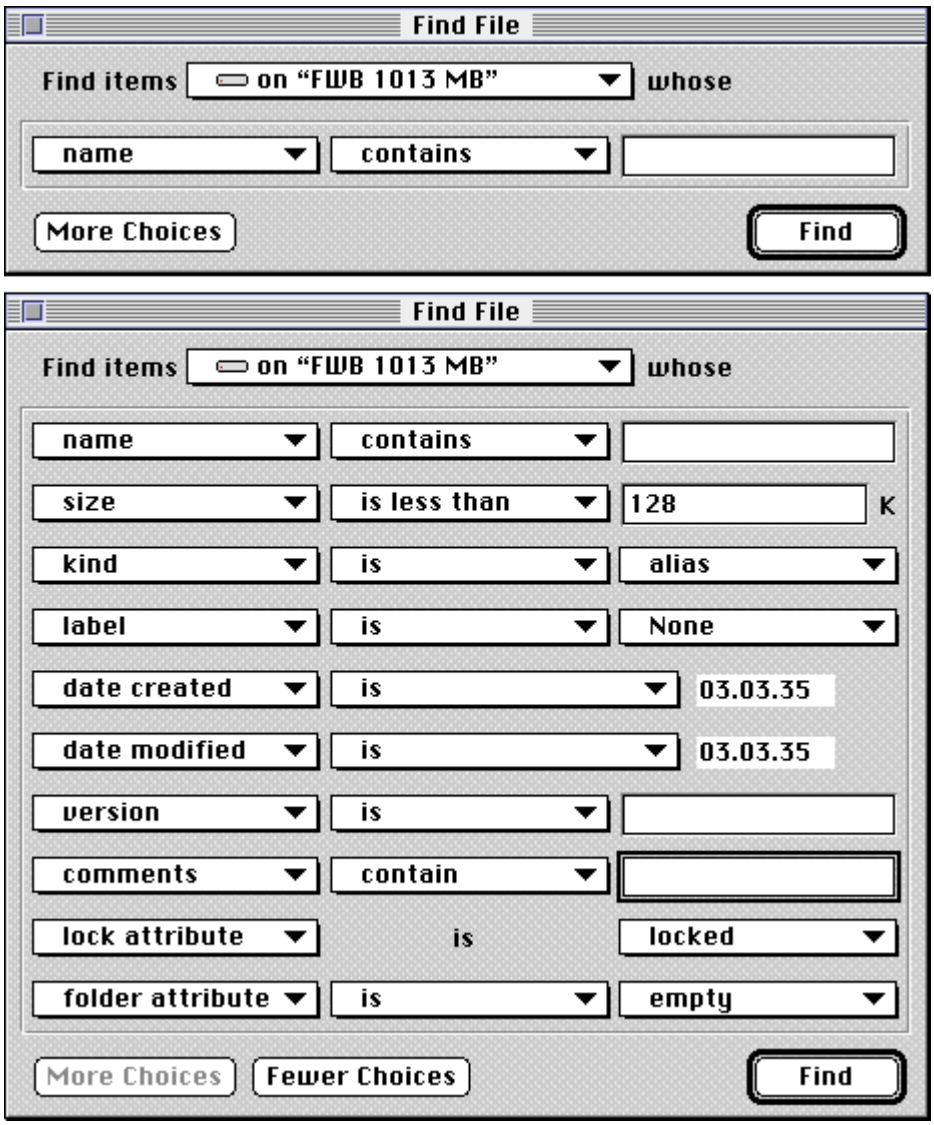


Рис. 53. Увеличивающееся диалоговое окно. Сверху начальное состояние окна, снизу – конечное. Нажатие на кнопку More Choices увеличивает окно. Это диалоговое окно интересно тем, что для открытия полного окна пользователям приходилось несколько раз нажимать на кнопку. Это плохо, поскольку, чтобы узнать о существовании дополнительных критериев поиска, пользователям приходилось совершать слишком много действий. Неудивительно, что в последних версиях MacOS эта утилита была полностью переработана. © Apple

Это и породило проблему. Раньше разные вкладки содержали примерно одинаково важные элементы, просто не все помещались в одно окно, а кнопка с треугольником скрывала элементы, про которые начинающий пользователь твердо знал, что они ему не нужны или пользоваться ими опасно. Теперь все во вкладках, поэтому пользователи часто уверены, что сразу невидное опасно.

В результате некоторые пользователи избегают пользоваться элементами, расположенными на изначально закрытых вкладках, даже если это ничем им не грозит. Поэтому нежелательно размещать на закрытых вкладках элементы, которые пользователям обязательно понадобятся, даже если эти элементы и не нужны постоянно (в этом случае правило про релевантность должно отступать). Разумеется, это не касается опытных пользователей.

В интернете и в остальных операционных системах, которым Microsoft была не указ, кнопки, увеличивающие размер окна и открывающие продвинутые элементы управления, сохранились в полном объеме. Учитывая тот

факт, что никаких пользовательских проблем с ними не замечено, можно смело рекомендовать их и для использования в Windows, тем более что они позволяют добиться определенного брэндинга.

Похоже, что Microsoft сама осознала вред от потери увеличивающихся окон, вызванный недоверием пользователей ко вкладкам. Во многих продуктах она стала использовать несколько усложненный элемент управления TreeView, также позволяющий запихнуть очень много элементов в ограниченное пространство. К сожалению, TreeView, если вместить в него дополнительные элементы управления, крайне неудобен в обращении[1], так что похвалить Microsoft за использование этого элемента затруднительно.

Число вкладок. Теоретически число вкладок может быть сколь угодно большим. На практике оно ограничивается двумя факторами: во-первых, объемом КВП (см. «Кратковременная память» на стр. 47), а во-вторых, размером области, в которые ярлыки вкладок могут помещаться. Дело в том, что если ширина всех ярлыков будет больше ширины окна, придется либо делать несколько строк ярлыков, либо скрывать часть из них, пока пользователь не нажмет специальную кнопку. И то и другое плохо.

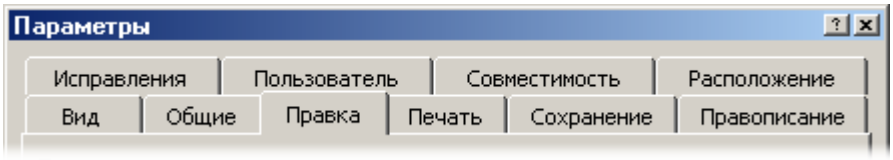


Рис. 54. Пример нескольких строк ярлыков. Если пользователь щелкает по ярлыку из верхнего ряда, весь этот ряд становится нижним, нижний же перемещается вверх. При этом две строки есть вовсе не предел – порой встречаются диалоговые окна с тремя рядами ярлыков. © Microsoft

Несколько строк ярлыков плохо по двум причинам. Во-первых, из-за большого количества мелких деталей (границ ярлыков), вся конструкция довольно медленно сканируется, т. е. трудно найти нужную вкладку. Во-вторых, при последовательном обращении к нескольким вкладкам из разных рядов эти ряды меняются местами, т. е. каждый раз нужно заново искать нужную вкладку. И то и другое крайне негативно сказываются на субъективном удовлетворении и скорости работы.

Рис. 55. Пример частично скрытых ярлыков вкладок. © Selenic Software

Скрывать часть ярлыков тоже нехорошо. Предположим, что пользователь нажал на стрелку вправо, показывающую следующую часть ярлыков. Если при этом значительно пролистывать строку с ярлыками, пользователи будут полностью потерять контекст (сильнее даже, чем они теряют его, нажимая Page Down). Если же пролистывать строку по одному элементу, контекст не потеряется, но перемещение между вкладками будет очень медленным.

Существует и третий способ решения проблемы – можно просто убрать вкладки, заменив их раскрывающимся списком. Этот способ тоже не слишком хорош, поскольку не слишком визуален и к тому же сравнительно медлителен.

1. Я не включил в книгу описание этого элемента именно потому, что имеющиеся TreeView неудобны и непонятны многим пользователям. Возможно, что они эволюционируют во что-то приличное, но пока правило просто: если вам нужна скорость работы или пользователи не имеют большого опыта, избегайте TreeView всеми силами.

Похоже, что самым эффективным решением является комбинация второго и третьего способов: основные экраны реализуются в форме вкладок, а дополнительные вызываются через раскрывающийся список. Это позволяет обеспечить максимальное количество наглядности и скорости работы.

Объем содержимого. Фактически, каждая вкладка представляет собой отдельное диалоговое окно внутри другого диалогового окна. Поэтому странной выглядят попытки (встречающиеся огорчительно часто) рассортировать элементы управления так, чтобы во всех вкладках их было поровну. Делать это ни в коем случае нельзя. Один экран должен содержать только те элементы, которые в нем нужны и которые пользователь может в этом экране ожидать.

Не старайтесь рассортировать элементы так, чтобы в каждой вкладке их был одинаковое количество

Терминационные кнопки. В диалоговом окне с вкладками терминационные кнопки обязательно должны располагаться вне области вкладок.

Перемещение в пределах окна

Помимо навигации между экранами, существует еще и навигация внутри отдельных экранов. Пользователям необходимо дать возможность максимально быстро переходить к необходимым элементам управления. Для этого у них есть два способа – мышь и клавиатура.

С мышью все более-менее понятно: закон Фитса (см. «Быстрый или точный» на стр. 10) работает на полную катушку. Поэтому оптимизация диалогового окна, уменьшающая дистанцию перемещения курсора, всегда приводит к росту (хотя и небольшому) производительности пользователей.

С клавиатурой сложнее. Пользователь может перемещаться между элементами управления двумя разными способами: клавишей Tab и горячими клавишами. Перемещаться клавишей Tab медленно, но зато для этого не нужно обращаться к памяти или высматривать клавиатурную комбинацию для нужного элемента. Напротив, горячие клавиши позволяют быстрее перемещаться вглубь экрана, но требуют запоминания клавиш. Таким образом, пользователи, которые часто вводят данные в какой-либо экран, стараются использовать клавишу Tab и только изредка пользуются горячими клавишами. Соответственно, любая форма ввода, которой часто пользуются, обязана корректно работать с Tab, при этом желательно, чтобы она работала и с горячими клавишами.

Проверяйте все диалоговые окна на отсутствие полей вывода, на которые устанавливается табуляция

Работа пользователей с клавишей Tab может быть омрачена по двум причинам. Во-первых, на экране могут быть элементы, не подразумевающие взаимодействия с пользователем (например, скрытая или заблокированная кнопка, поле вывода), но на которые перемещение совершается. Избавиться от этой проблемы легко – нужно лишь явно указать, чтобы в список объектов, между которыми можно перемещаться, ОС их не вносила. Во-вторых, бывают ситуации, когда визуальный порядок элементов управления (происходящий из-за того, что пользователи читают экраны) не совпадает с порядком перемещения. В этом случае нужно просто сменить у неправильных элементов их место в последовательности.

Особым вариантом окон являются действия, выполняющиеся в последовательности сменяющих друг друга окон (мастера). Чтобы осознать правила, применимые к ним, полезно определить причины, вызвавшие появление таких окон.

Рис. 56. Пример мастера.

Во-первых, существуют действия, для которых либо естественна, либо желательна жесткая последовательность (см. «Определение смысловой связи между блоками» на стр. 118). Для таких действий единый экран, в котором выполняется вся последовательность, не слишком эффективен: он грозит человеческими ошибками, к тому же, чтобы его использовать, требуется построить ментальную модель экрана (чтобы, как минимум, знать, что нужно сделать в начале, а что в конце). Эффективнее разбить действие на несколько разных экранов.

Во-вторых, существуют действия, которые всегда будут вызывать проблемы у пользователей, либо потому, что эти действия сложны, либо потому, что нужны они редко (так что пользователям нет резона учиться). При этом единое окно для выполнения действия также оказывается неэффективным, поскольку объем справочной информации, которую в него нужно вместить, слишком велик. В таких случаях разделение действия на последовательность экранов позволяет снизить насыщенность отдельных экранов и тем самым освободить место для справочной информации.

Как правило, одной причины достаточно, чтобы оправдать использование мастера, когда же действуют обе причины, мастер становится обязательным. Итак, теперь, когда определены причины возникновения мастеров, можно перейти к конкретным правилам их создания.

Переход между экранами. Понятно, что пользователи должны получить возможность переходить не только на следующее окно в последовательности, но и на предыдущие окна. Менее очевидным является другое требование к мастерам: переход должен быть максимально легким.

Задача раскладывается на две составляющие: во-первых, нужно реализовать возможность свободного перемещения по последовательности. Если экранов немного (3-5), то вполне можно ограничиться стандартными

кнопками Назад и Далее. Если же экранов много, переход по этим кнопкам будет, как минимум, медленным. В таких случаях разумно дополнять кнопки раскрывающимся списком (при этом, возможно, исключая из него ещё не пройденные экраны), либо, если есть место, снабжать мастера списком всех экранов (отмечая текущий и пройденные экраны). Независимо от числа экранов в последовательности, необходимо информировать пользователей об объеме оставшегося действия (чтобы дать им возможность оценивать количество работы и тем самым повышать их чувство контроля над системой). Справедливости ради надо уточнить, что в длинных последовательностях показ объема оставшихся экранов может снизить мотивацию пользователей в начале действия, но повысить мотивацию в конце («осталось немного, не буду это бросать»).

Вторая составляющая – четкость перехода. Для пользователей мастер, т. е. последовательность экранов, кажется единым экраном, содержимое которого меняется. Эту иллюзию нужно поддерживать, поскольку она позволяет не сбивать контекст действий пользователя и поддерживать внимание на «сюжетно-важной» области экрана. Для этого размер и расположение окна мастера, а также расположение и облик всех повторяющихся элементов (таких как терминационные кнопки) нужно выдерживать неизменными на протяжении всей последовательности.

Контекст. В отличие от единого окна, в котором выполняется действие, в мастерах необходимо поддерживать контекст действий пользователя. Поскольку ранее сделанная работа скрыта, пользователи могут потерять контекст, что может замедлить действие (контекст придется восстанавливать). Степень потери контекста зависит от количества экранов, времени, которое пользователи проводят за отдельными экранами и от времени реакции системы. И если количество экранов в мастере редко превышает шести (а это небольшое число), то время, проведенное на пройденных экранах и, особенно, реакция системы (особенно в интернете), могут быть значительными.

Единственным же средством поддержания контекста является вывод текущего состояния данных в процессе выполнения мастера. Как правило, обычный текстовый список с предыдущими установленными параметрами работает плохо (к тому же редко вмещается в экран), визуализировать же изменения трудно, если вообще возможно. Таким образом, лучше избегать длинных последовательностей (тем более что уровень мотивации пользователей при увеличении продолжительности действия снижается).

Вывод справочной информации. Благодаря обилию пустого места мастера замечательно подходят к выводу справочной информации в самом интерфейсе (см. «Что нам нужно и что у нас есть» на стр. 30). Справочную же информацию нужно выводить двух типов, а именно краткое и более развернутое описания текущего шага.

С развернутым описанием все просто. Где-нибудь снизу экрана (чтобы не сбивать фокус внимания пользователей) выводится один или два абзаца, рассказывающие стандартный набор: что, зачем и почему. С кратким же описанием сложнее. К сожалению, устоявшийся облик мастеров не имеет большого и заметного заголовка (этой проблемы, к счастью, нет в интернете, где вообще нет ничего устоявшегося). Например, даже такой прогрессивный мастер, как использованный в примере, не имеет явно видимого заголовка. Шрифт мелкий, к тому же его разборчивость ослаблена фоном. Это неправильно. У каждого окна последовательности должен быть ясно видимый и бросающийся в глаза заголовок. При этом, в отличие от обычных заголовков окна, он должен быть написан не описательно, но командно (сделайте то-то и то-то). Microsoft, в некоторых своих продуктах широко использующая мастера (называя это побуждающим пользовательским интерфейсом) вообще рекомендует считать заголовки важнейшими элементами мастеров. Особо подчеркивается, что заголовки экранов должны быть созданы и сформулированы до начала проектирования экранов, при этом содержимое экранов не должно выходить за рамки смысла заголовков. Поспорить с Microsoft в данном случае затруднительно.

Остальное

Помимо элементов управления, меню и окон, интерфейс состоит из многих других «кирпичей», основными из которых являются пиктограммы, курсоры, цветовое кодирование и звуки.

Пиктограммы

Создание пиктограмм является для дизайнера весьма приятной работой. К сожалению, работа эта порой оказывается либо бесполезной, либо ненужной. Дело в том, что польза и возможности пиктограмм очень сильно переоценены.

Чем пиктограммы плохи

В 1981 году Истерби и Грейдон провели масштабное исследование[1] ста восьми пиктограмм, выбранных экспертами ISO для использования по всему миру. Некоторые из этих пиктограмм широко использовались и до этого. Цель исследования заключалась в том, чтобы определить, сколько пиктограмм правильно бы распознавались двумя третями целевой аудитории. Результат: три[2].

Собственно говоря, это и есть основной недостаток пиктограмм: чаще всего, чтобы распознавать их смысл, нужно сначала выучить их значение. Вера в то, что пиктограммы могут нести точное и понятное всем значение, практически лишена оснований.

Но постойте, скажут мне, ведь пиктограммы всё-таки как-то распознаются пользователями. И тут я не смогу не согласиться. Но все дело в том, что распознаются они не благодаря своим достоинствам. Разберем это подробнее.

Ученые обнаружили, что (в зависимости от языка и от персональных предпочтений говорящего) слова в разговоре передают только 10-20 процентов общего смысла речи. Остальное передается интонацией, паузами, жестикуляцией и т.д. Примерно та же ситуация и с пиктограммами, роль же интонации и прочего выполняет *контекст*, в котором эта пиктограмма располагается.

Рис. 57. За пять минут я нашел в MS Word четыре совершенно непонятных пиктограммы (не считая множества слабо понятных). В то же время в контексте, т. е. в соответствующих диалоговых окнах, эти пиктограммы значительно более понятны.

Дело в том, что понимание того, что именно на пиктограмме изображено, не приводит автоматически к пониманию того, что эта пиктограмма означает. Например, первая пиктограмма на иллюстрации, возможно, обозначает одновременное подчеркивание и надчеркивание литер текста.

1. Easterby, R. S., & Graydon, I.R. Evaluation of public information symbols, ISO tests: 1979/80 series, part II: comprehension/recognition tests. (AP Report 100). Birmingham, England: Applied Psychology Department, University of Aston, 1981.
2. Недаром все пиктографические системы записи были вытеснены буквенными алфавитами.

А возможно, что и нет (собственного говоря, нет). Или другая пиктограмма, с буквой «А» и стрелками. Сказать, что именно она делает, невозможно. Еще сильнее этот эффект проявляется в третьей и четвертой пиктограммах. На них изображены предметы столь многозначительные, что диапазон значений, которые в них можно вложить, становится поистине безбрежным.

Происходит это оттого, что размер пиктограммы практически не позволяет нарисовать на ней что-либо однозначное. А если размер увеличить, то пиктограмма будет пожирать слишком много места и очень долго распознаваться пользователями (о чем ниже). При этом понимание смысла неоднозначности возможно только при наличии контекста. Например, всем понятно, что четвертая пиктограмма в иллюстрации, замок, обозначает, что нечто запирается. Но что именно запирается и как именно запирается, сказать невозможно. Однако если пиктограмма расположена в диалоговом окне, становится понятно – запирается всё, что находится в окне; если пиктограмма в рамке группировки – запирается всё, что находится в рамке; если на другой пиктограмме – запирается объект, который данная пиктограмма символизирует.

Таким образом, смысл пиктограммы передаётся не столько пиктограммой, сколько контекстом. В этом нет ничего ужасного. Плохо то, что зачастую невозможно нарисовать даже что-либо простое и однозначное. 32 на 32 пикселя – это очень мало, а, учитывая еще и суровые требования к ракурсу, получается ещё меньше. При этом не надо забывать, что в реальной жизни и 32 на 32 пикселя есть очень много; зачастую максимальным возможным размером является 16 на 16.

Однако у пиктограмм есть и другая проблема, точнее, более глубокая сторона первой проблемы. Как уже было сказано, понимание того, что именно на пиктограмме изображено, не приводит автоматически к пониманию того, что она означает. Так вот, порой оказывается, что пользователи не понимают даже, что изображено на пиктограмме. В этом нет ничего исключительного: если рисовать *редкий* предмет реального мира, ситуация будет именно такой. При этом такая непонятая пиктограмма превращается, фактически, в кучку по-разному окрашенных пикселей. Это значит, что сюжетом пиктограммы может быть только нечто либо очень простое (ежедневник, замок), либо составленное из того же очень простого (замок, из которого вынимается или вставляется ключ). Это еще более ограничивает возможности.

Таким образом, чтобы пользователи могли правильно распознать пиктограмму, пиктограмма в идеальном случае должна обладать следующими свойствами: во-первых, она должна иметь внятный и однозначный контекст, во-вторых, её сюжет должен быть пользователям понятен, в-третьих, этот сюжет должен быть удовлетворительно нарисован, что удаётся отнюдь не всегда. Другой вариант, рассчитанный не на идеальный случай: чтобы пользователи могли правильно распознать пиктограмму, они должны узнать её значение и запомнить его для использования в будущем. По-моему, этот вариант гораздо реалистичней. При этом от пиктограммы не требуется ни особой внятности, ни контекста, ни красот реализации. Единственными требованиями являются степень отличия пиктограммы от других и количество легко запоминающихся свойств, таких как синий прямоугольник в левом углу, множество красного цвета или изображение ножниц.

Суммируя, можно сказать, что пиктограммы практически не делают интерфейс более понятным и легким в обучении. Именно это несоответствие желаемого и действительного и составляет главный недостаток пиктограмм.

Чем пиктограммы хороши

Достоинств у пиктограмм несколько. Во-первых, они увеличивают скорость поиска элементов взглядом за счет того, что этот элемент с пиктограммой становится более заметен. Однако и здесь пиктограммы не без греха: чтобы скорость поиска увеличилась, не все элементы должны быть снабжены пиктограммами, а только некоторые; в идеале иметь пиктограмму должен только нужный элемент. Во-вторых, они служат хорошими индикаторами важности элементов, поскольку элементы с пиктограммами всегда воспринимаются как более важные, по сравнению с теми элементами, у которых пиктограмм нет. В-третьих, в определенных ситуациях они действительно ускоряют скорость обучения. Но об этом отдельно.

Пиктограмма есть непосредственное манипулирование для бедных

Существует определенный набор пиктограмм, ставших настолько стандартными, что воспринимаются мгновенно и легко. Так, пиктограмма, изображающая домик с трубой, стала настолько популярной кнопкой перехода на титульную страницу сайта, что пользователям гораздо легче воспринимать эту пиктограмму, нежели чисто словесную кнопку. Обратите внимание, что такой невольной стандартизации подвергается не только сюжет пиктограммы, но комбинация сюжет/значение; по отдельности они интереса не представляют.

Такие пиктограммы, безусловно, составляют «золотой фонд» дизайнера интерфейсов, поскольку их использование не несет никаких проблем, зато сулит существенное упрощение жизни. К сожалению, и здесь не всё безоблачно: во-первых, большинство дизайнеров считает такие пиктограммы необычайно скучными и набившими оскомину, во-вторых, таких пиктограмм очень немного[1]. И если со скучностью ещё можно как-то бороться, то с недостаточным количеством пиктограмм ничего не поделать.

Место пиктограммы в жизни

В традиционном ПО пиктограмм используется довольно много:

- пиктограмма системы
- пиктограммы отдельных файлов системы, которые пользователи могут как-либо изменять
- пиктограммы документов, которые создаются и редактируются системой
- пиктограммы инструментов системы (не всегда, многие системы отлично живут и без них)
- пиктограммы панели инструментов и меню
- пиктограммы объектов, использующихся для реализации непосредственного манипулирования (изредка, к сожалению).

Поскольку интернет сейчас находится в зачаточном состоянии (по сравнению с индустрией ПО), количество типов пиктограмм в нем фактически исчерпывается одной позицией (исключения бывают, но они именно исключения):

- пиктограммы меню.

В целом, список применений пиктограмм в ПО достаточно хорош и убедителен, хотя в нём имеется определенный перекос в сторону пиктограмм файловой системы за счет пиктограмм объектов. Впрочем, это объясняется сугубо эволюционными причинами: для большинства разработчиков ПО сама идея непосредственного манипулирования стала сколько-нибудь популярна совсем недавно, так что перелом ещё не наступил.

1. Малый объем кодекса стандартных пиктограмм зачастую играет с дизайнерами злую шутку. Так, в Интернете есть стандартные пиктограммы для перехода на титульную страницу (домик) и для отправки почты владельцам сайта (конверт). А вот столь же стандартной пиктограммы для перехода на карту сайта нету, отчего, ради соблюдения единообразия, часто приходится лишать пиктограмм и кнопки, у которых всё с пиктограммами хорошо.

Список мест пиктограмм в системе, приведенный в предыдущей главе, нужен был преимущественно для того, чтобы разграничить универсальные требования к пиктограммам, применяемые ко всем типам, и не универсальные, применяемые к отдельным типам. Соответственно, сначала универсальные требования.

Разборчивость. Под разборчивостью я понимаю намеренное усиление контраста в изображении. Такое усиление вообще характерно для экранной графики, но особенно для пиктограмм. Предназначено оно для того, чтобы максимально усилить уникальность формы пиктограммы и тем самым увеличить степень её запоминаемости. Также, в понятие разборчивости входит требование непохожести пиктограмм друг на друга.

Стандартность сюжета и его реализации. Пиктограмма домика в Web исторически обозначает переход на титульную страницу сайта. Как уже говорилось, благодаря этому пользователю не нужно проявлять излишнюю мыслительную активность при определении значения такого элемента (вообще говоря, человеческий мозг наиболее успешно решает задачи, метод решения которых основан на проведении аналогии). Важна также стандартность реализации выбранного сюжета, например, я, однажды видел пиктограмму перехода на титульную страницу, выполненную как изображение избушки на курьих ножках (с одной стороны, это всё-таки дом, с другой стороны – дом нестандартный). В таких условиях наилучшим методом выбора сюжета является поиск репрезентации, привычной целевой аудитории. Источниками стандартных сюжетов/реализаций являются (отсортированы по степени знакомства с ними пользователей): операционная система, среда, системы конкурентов. Таким образом, сюжет, взятый из операционной системы всегда оптимален (поскольку наиболее знаком). Некоторые понятия имеют однозначные визуальные репрезентации только для какой-либо одной аудитории, так что если эта аудитория является целевой, необходимо проверять качество восприятия символа на представителях именно этой аудитории (понятность или непонятность сюжета для всех остальных в таких случаях не может служить критерием эффективности).

Пиктограмма с текстом внутри картинки нехороша. Либо текст, либо изображение

Минимально возможная детализация сюжета. Объекты реального мира перегружены мелкими деталями. Перенос этих деталей в сюжет пиктограммы неоправдан – в реальном мире эти детали нужны (придают объекту уникальность), а на пиктограмме лишь отвлекают внимание и тем самым замедляют распознавание. Более того, скорость восприятия чистых абстракций чаще всего выше скорости восприятия даже максимально упрощенных символов, т. е. символ человеческого лица, нарисованный по принципу «круг, две точки и две черточки» воспринимается быстрее и легче, нежели тот же символ, нарисованный, например, в стилистике комиксов.

Направление теней во всех пиктограммах должно быть одинаковым: снизу справа

Стандартность стилистики. Все пиктограммы в системе должны обладать единой стилистикой. Под стилистикой я понимаю последовательное выдерживание единого ракурса и иных изобразительных приемов.

Рис. 58. Различие в стилистике на примере пиктограммы файла неопределенного документа. © Microsoft, Apple, Be, Sony.

Стилистика может быть как внешней, позаимствованной, например, у Microsoft, так и внутренней, уникальной. Заимствованная стилистика хороша стандартностью, о пользе которой сказано достаточно, уникальная же хороша своими возможностями брэндинга. В целом, если организация производит несколько сколько-нибудь пересекающихся систем, лучше выбрать брэндинг.

Эстетическая привлекательность. По большей части субъективна.

Полнота набора. Как знает любой программист и редкий веб-дизайнер[1], одна и та же по смыслу пиктограмма должна быть продублирована в системе в нескольких разных вариантах, поскольку пиктограмма может проявляться в различных контекстах. Так, главная пиктограмма программы в Windows проявляется не только в Explorer, но и на панели задач, в строке названия программы и в некоторых других местах. Проблема в том, что основная пиктограмма имеет размер 32 на 32 пикселя, но почти везде она должна показываться с половинным размером (16x16 пикселей). Если не создать дополнительно уменьшенную пиктограмму, то основная пиктограмма будет просто уменьшаться при выводе, что существенно, вплоть до полной неразборчивости, портит её качество (см. рис. 59). Подробнее наборы пиктограмм будут разобраны позже, пока достаточно сказать, что чем полнее набор, тем лучше.

Рис. 59. Пример набора пиктограмм. Слева основная пиктограмма MS Word и пиктограмма его документа. Если не нарисовать уменьшенных пиктограмм, то в строке названия программы и в меню будет показываться нечто непонятное (в центре). Однако если уменьшенные пиктограммы нарисовать отдельно, никаких проблем не возникнет (справа).

Правило формирования набора очень простое: все пиктограммы в наборе должны максимально походить друг на друга. К сожалению, простота это правила не приводит автоматически к его повсеместной выполнимости; если с парой 32x32 и 48x48 (о которой позже) проблем никогда не возникает, то с парой 16x16 и 32x32 проблемы происходят постоянно, поскольку красота большой пиктограммы не влезает в малую. Эта проблема имеет два решения: во-первых, можно *сначала* рисовать маленькую пиктограмму, а уже на её основе рисовать большую, во-вторых, можно выделить в большой пиктограмме основной элемент, и перенести в малую только его (пример такого подхода см. рис. 59). Первый вариант обеспечивает большее сходство пиктограмм, а значит, большую узнаваемость, зато второй значительно проще и потенциально более гибкий.

Пиктограмма системы и пиктограммы файлов. Несмотря на то, что владельцы всех операционных систем пытались и пытаются заставить разработчиков придерживаться определенных правил при создании основных пиктограмм системы, только очень небольшая часть разработ-

1. Веб-дизайнерам это пока неактуально.

чиков их послушалась, и поступила неправильно. Столь незначительная возможность установить последовательность в реализации интерфейса не стоит потери брэндинга.

Иная ситуация с пиктограммами документов. Здесь сами разработчики чувствовали пользу от стандартизации и были согласны с предложенным правилом. Тем более что правило у всех операционных систем одинаковое и очень простое.

Пиктограмма документа образуется от суммирования пиктограммы неопределенного документа из ОС и мотивов, почерпнутых из основной пиктограммы системы

Это правило хорошо двумя вещами: во-первых, оно очень гибкое, поскольку не говорит, какие мотивы основной пиктограммы нужно брать, а во-вторых, оно очень жесткое, поскольку пиктограмма неопределенного документа есть императив. Хороший пример пиктограммы документа приведен в предыдущей иллюстрации.

Подобным образом создаются пиктограммы шаблонных файлов: пиктограмма шаблонного файла образуется от суммирования двух или трёх, положенных друг на друга, пиктограмм неопределенного документа и мотивов, почерпнутых из основной пиктограммы системы.

Рис. 60. Примеры пиктограмм для файлов шаблонов. Первые три демонстрируют стиль Microsoft, последние две – старый стиль Apple.

И опять правило повторяется (слегка модифицированное). Для служебных файлов, которые, однако, нужны не только системе, но некоторым пользователям, тоже нужно делать пиктограммы. Для них правило звучит следующим образом: пиктограмма служебного файла образуется от суммирования пиктограммы неопределенного документа из ОС, мотивов, почерпнутых из основной пиктограммы системы, и символа служебности файла. Символы служебности файла различаются в разных ОС, так, в Windows это две шестеренки, а в Mac OS их несколько (на все случаи жизни).

Теперь о наборе пиктограмм. Поскольку пиктограммы этих типов показываются не только в самой системе, но и в ОС, делать их необходимо в трех вариантах (подразумевается, что они делаются для Windows): стандартного размера (32x32), уменьшенного (16x16) и, поскольку в последнее время мониторы выиграли в разрешении, но не выиграли в размере, увеличенного (48x48).

Пиктограммы панели инструментов и меню. Поскольку пиктограммы панели инструментов и меню выводятся всегда малого размера (16x16, в интернете бывает, что и меньше), они вызывают обычно максимум проблем, тем более что их обычно очень много. Понятно, что в таких условиях не до трехмерности и ракурсов, чаще всего приходится удовлетворяться сравнительно убогими вариантами. С другой стороны, не приходится мучиться, рисуя полный набор.

Трудность создания таких пиктограмм не позволяет выводить какие-либо жесткие правила (мы не мазохисты), тем не менее, определенная тонкость есть и здесь. В последнее время Microsoft ввела моду делать пиктограммы на панели инструментов серыми, включая в них цвет только в моменты, когда курсор находится над пиктограммами. Конечно, эта мода хороша тем, что панель инструментов перестает походить на новогоднюю ёлку (особенно слявятся этим продукты компании Lotus), но, с другой стороны, при этом снижается разница между отдельными пиктограммами, т. е. снижается раз-

борчивость. При старой моде, равнодушно относящейся к цвету, если одна пиктограмма в ряду содержала много синего цвета, она отличалась от остальных, не синих. Серая панель инструментов, к сожалению, такого достоинства не имеет. Так что и старая мода и новая имеют определенные достоинства: выбирать вам.

Обратите внимание, что этот раздел не заменяет, а дополняет сведения, которые можно почерпнуть из руководств разработчика, публикуемых создателями операционных систем.

Курсоры

Курсоры есть замечательное средство обеспечения обратной связи. Так, всякий раз, когда пользователь подводит курсор к углу окна, курсор изменяется, показывая пользователю, что форму окна можно увеличить. К сожалению, у курсоров есть и обратная сторона: пользователи способны примириться с некрасивой пиктограммой, поскольку они могут не смотреть на неё, примириться же с некрасивым курсором они не могут, потому что они вынуждены смотреть на него всё время, пока он присутствует на экране.

Поэтому правило использования курсоров звучит следующим образом: используйте индикацию видом курсора во всех случаях, когда вы можете сделать это эстетически привлекательно; в остальных случаях о курсорах даже и не думайте.

Но что вообще входит в понятие «красивого курсора»? Это, по преимуществу, все внешние курсоры, неважно, откуда они приходят, из ОС для ПО или CSS для сайта (набор курсоров в обоих источниках практически совпадают). Можно, конечно, нарисовать и собственные курсоры, но я не рекомендую это делать, поскольку все хорошие курсоры уже были кем-то нарисованы, так что вы можете легко нарушить чьи-то авторские права. Если же вы всё-таки решитесь рисовать собственные курсоры, старайтесь придерживаться двух правил: избегайте цвета и делайте курсоры такими, чтобы активная точка курсора (которая указывает) была возможно ближе к верхнему левому краю курсора.

Цвет

Пожалуй, ни одна тема не вызывает у дизайнеров такого интереса и воодушевления, как цвет. Извечные споры «какой колёр лучше» не утихают и не утихнут, похоже, никогда. Однако в рамках этой книги нет места для этих споров, так что мы ограничимся малым.

Во-первых, следует сразу сказать, что цветами невозможно передавать пользователям сколько-нибудь сложную информацию. Как правило, схемы типа «этот объект тёмный, значит...» не работают, поскольку требуют от пользователей длительной тренировки. С другой, стороны, люди хорошо способны замечать и анализировать взаимосвязи цветов, так, строки в таблице, окрашенные в разные цвета, подсознательно будут группироваться пользователями по цветовому признаку. Обратите внимание, что работать это будет только при очень ограниченном числе используемых цветов (как можно догадаться, максимальное число цветов равно объему кратковременной памяти). Однако на практике, всё, что можно передать цветом, умещается в одно предложение: для всех красный цвет обозначает «нельзя» или «осторожно», для многих людей зеленый обозначает «поехали» (от умения пользоваться светофором; интересно, что желтый цвет не работает, хотя должен). Реально актуальна только первая половина предложения, поскольку в нашем деле фраза «для многих людей» часто значит «ни для кого».

Не используйте кодирование цветом без крайней необходимости, когда же вы всё-таки его используете, добавляйте дополнительные способы индикации

Во-вторых, восприятие цвета индивидуально, при этом порой слишком сильно. Я не имею в виду дальтоников (хотя и их тоже), просто достаточно большое количество людей использует старые мониторы, на которых цвета «поехали», и, что более важно, с возрастом зрение слабеет, при этом видимые цвета темнеют и выцветают. Это значит, что нельзя использовать цвет как *единственный* индикатор чего-либо, обязательно должен быть и другой признак. Из-за этой индивидуальности восприятия цветов происходит другое правило: позволяйте пользователю самому выбрать цвета, поскольку цвет, нравящийся одному человеку, возможно, будет вызывать омерзение у другого (желательно делать это системными средствами).

Звук

Звук, как и цвет, в интерфейсе не есть вещь абсолютно необходимая. Скажем так, он практически не может делать что-либо полезное, помимо индицирования конца какого-либо процесса (благодаря которому пользователь освобождается от необходимости поминутно поглядывать на экран). Это происходит вовсе не потому, что звук плох, но лишь потому, что очень незначительное количество компьютеров имеет работающие звуковые подсистемы. Только в играх можно рассчитывать на наличие звука, исходящего из компьютера пользователя. Таким образом, звука мы оказываемся лишены. В то же время и о звуке есть что сказать, хотя и не много.

Система, завершив какую-либо длительную операцию, должна пищать

Во-первых, восприятие звуков у каждого человека уникально ещё более чем даже восприятие цвета. Если пользователям *полезно* давать возможность изменить цвета системы, то применительно к звукам правило звучит более жестко: *обязательно* позволяйте пользователю самому выбирать звуки (и уж как минимум отключать их вовсе). Во-вторых, возможность издавать полноценные звуки есть отнюдь не у каждого компьютера, в особенности это касается офисных машин: у кого-то нет звуковой платы, кому-то запрещают включать звук, чтобы не раздражать соседей по комнате. Это значит, что издавать писк при окончании операции нужно не с помощью звуковой платы, а с помощью зуммера, который есть в каждом компьютере. В-третьих, надо всегда помнить, что качественный и приятный фоновый звук чаще всего не воспринимается пользователями как благо, но зато при этом звуке субъективно улучшается *изображение* (процентов на 20). Это и есть главный аргумент за хороший звук.

Часть iii

Процесс

В этой части описана методология дизайна интерфейса, обеспечивающая достаточное качество работы, её простоту, и, что немаловажно, соблюдение сроков её выполнения.

Три шага к совершенству

В процессе дизайна интерфейса выделяются три основных этапа, а именно первоначальное проектирование (часто оказывающееся и окончательным), создание прототипа и тестирование/модификация прототипа.

Фактически процесс дизайна, чтобы быть успешным и безусловным, всегда стремится происходить в этой последовательности: проектирование, затем создание прототипа, затем тестирование/модификация, затем опять тестирование/модификация, затем опять тестирование/модификация, затем опять тестирование/модификация, затем нечеловеческие крики начальства «Хоре волынку тянуть, шоб через пять минут усё было!». Т. е. основным этапом оказывается не дизайн, но полировка уже сделанного дизайна. С другой стороны, при тщательном проектировании длительного тестирования обычно удается избежать – но, с другой стороны, при этом проектирование становится достаточно длительным, так что неизвестно еще, что лучше сокращать.

Этап проектирования сам по себе состоит из нескольких составляющих, причем количество этих составляющих довольно велико. Радует, однако, то, что существует очень мало работ, при которых нужно выполнять *все* составляющие (собственно говоря, если выполнять всё-таки придется, это должны делать разные специалисты). Поскольку объем книги ограничен, здесь описываются только самые необходимые.

При этом важно понимать, что здесь описываются только методы создания *новой* системы. Модернизация старой требует совершенно других методов и усилий, поскольку при этом приходится еще и безболезненно исправлять имеющиеся недостатки (что чаще всего вообще невозможно по организационным причинам). Как это ни печально, но в настоящее время не найдено сколько-нибудь универсального метода радикального улучшения существующих систем. Зачастую вместо полноценного дизайна приходится удовлетворяться наведением общего глянца, что, в лучшем случае, придает системе сомнительный шарм молодящегося старика.

В то же время методология проектирования описана в литературе слабо (основной момент уделяется менеджменту, что в нашей стране не очень актуально). Объясняется это тем, что общие принципы еще не выработаны (за исключением отраслевых стандартов военных ведомств, которые нечасто познавательны и всегда неприемлемы для чтения). В результате все пользуются самодельными методами работы, в чём, по сути, нет ничего плохого (трудности появляются при синхронизации работы нескольких человек). Так что приводимые мной рекомендации преимущественно базируются на моем личном опыте и пристрастиях. С другой стороны, они имеют то несомненное достоинство, что сам я пытался ничего не придумывать, но брать готовое, что внушает надежду, что мои методы *могут* быть использованы кем-то кроме меня.

Необходимо также отметить еще и следующее. В процессе проектирования вам понадобится незаслуженно забытые инструменты – а именно ручка и бумага. Дело в том, что использование компьютера само по себе дело не быстрое, во-первых, поскольку интерфейс программ несовершенен, а во-вторых, из-за того, что, используя компьютер, вы будете подсознательно стараться сделать работу *красиво*, а не просто будете фиксировать свою мысль. Известно, например, что текстовые процессоры не принесли ускорения письма – человек, пишущий на компьютере тратит

уйму времени на полировку фраз. Владелец же печатающей машинки полирует фразы в голове, что гораздо быстрее. Более того, почти постоянно приходится делать много вариантов одного и того же – так зачем полировать до блеска вариант, который вы отбросите через пять минут? Тем более что после получения устраивающего результата всегда можно перенести его в компьютер. Бумага имеет еще и то преимущество, что её можно с успехом показывать руководству в качестве доказательств своей активности. Практика показывает, что вид сколотых скрепкой мятых исчерканных листов почти всегда производит на руководящий состав исключительное впечатление (мы-то с вами знаем, что внутрь этой пачки для толщины можно класть всё что угодно).

Множество перевернувших мир идей начали свою жизнь нарисованными на салфетке, что не никак не помешало им в дальнейшем. Так что запаситесь бумагой и вы. Понадобится её немало.

Роль технического писателя

Первым человеком, которого плохой интерфейс приводит в трепет, является отнюдь не конечный пользователь, но технический писатель – если интерфейс понятен, от писателя требуется мало работы, если непонятен – много. Это делает технических писателей лучшими друзьями дизайнеров интерфейсов.

Дело в том, что множество систем ни при каких обстоятельствах не могут быть используемы без подготовки, независимо от качества их интерфейса. Система автоматизации, например, может быть эффективно использована только в том случае, когда пользователь этой системы понимает *суть* автоматизируемых процессов. Обязанность же технического писателя, прежде всего, заключается в том, чтобы снабдить пользователя этим пониманием. Это значит, что концепции и суть сложной системы могут быть безболезненно вынесены из интерфейса в документацию, освобождая ресурсы дизайнера. С другой стороны, зачастую *описание* концепций влияет на их *реализацию* в системе, особенно в ситуациях, когда скорость обучения работе с системой является важнейшей составляющей качества. Это наблюдение вполне подтверждается опытом. Например, Джеф Раскин, Отец Макинтоша, изначально был начальником отдела документации. После того, как он обнаружил, что имеющуюся систему невозможно приемлемо описать, он создал новую, описывающуюся хорошо. Побочным свойством новой системы, компьютера Макинтош, было то, что его интерфейс был прост и удобен в работе.

Документация есть часть интерфейса, причем в сложных системах – большая часть

С практической точки зрения это значит, что технический писатель почти всегда знает о конструируемой системе не меньше, чем дизайнер, или, во всяком случае, знает *другое*. Это делает необходимость коммуникации дизайнера с писателем насущной необходимостью, более того, во многих источниках напрямую рекомендуется ставить писателя выше дизайнера, поскольку его вклад в проект более значителен.

Проблема этого подхода заключается в том, что роль документации в нашей стране чрезвычайно недооценена. Создание документации воспринимается как сравнительно необязательный процесс, нужный исключительно для утяжеления коробки с продуктом. От этого документация почти всегда пишется после того, как система создана, более того, писателям мало платят. В результате работа технического писателя не приносит ни денег, ни удовлетворения. Неудивительно, что средний уровень отечественных техписов, мягко говоря, невысок, что не позволяет рассчитывать на какую-либо помощь от них.

С другой стороны, хорошие технические писатели есть. И если вы нашли такого, смело считайте, что ваша работа будет легче и получится качественней. А если не нашли – ищите, не покладая рук.

Первоначальное проектирование

Важность этого этапа трудно переоценить. На нем закладываются основные концепции системы, влияющие абсолютно на все показатели качества её интерфейса. Как будет описано в главе о тестировании, структурные проблемы практически не могут быть обнаружены и решены на остальных этапах (для их обнаружения нужно слишком много везения, а для исправления – денег). Это значит, что чем больше внимания будет уделено проектированию, тем выше будет общее качество.

Собственно проектирование состоит из следующих этапов:

1 определение необходимой функциональности системы
2 создание пользовательских сценариев
3 прилив вдохновения
4 проектирование общей структуры
5 конструирование отдельных блоков
6 создание глоссария
7 сбор и начальная проверка полной схемы системы.

Каждый последующий этап в такой системе зависит от результатов предыдущих этапов. Соответственно, пропуск какого-либо этапа (за исключением, разве что, создания глоссария) негативно влияет на результаты всех последующих этапов.

Определение необходимой функциональности системы

На первом этапе необходимо определить функциональность будущей системы. Это исключительно важный этап, поскольку именно функциональность будет определять весь интерфейс.

Очень важно сознавать, что практически невозможно убрать из уже продающейся системы какие-либо функции. Программы до сих пор продаются по функциям (чем больше список функций на коробке с программой, тем легче её продать, даже если большинство из них либо не работает, либо не нужно пользователям). Происходит это потому, что существенную часть тиража программы покупают новые пользователи, которые ничего не знают про истинное положение вещей. Кроме того, имеющиеся пользователи обычно с исключительной неохотой переучиваются для использования новых функций взамен прежних (при этом еще и обижаются из-за того, что их инвестиции в обучение оказались выброшенными на ветер). Это значит, что ненужная функция системы будет кочевать из версии в версию, раздувая размеры программы, снижая надежность и быстродействие, и, что для нас более важно, портя интерфейс (при этом и длительность разработки возрастает). Несколько лучшая ситуация с сайтами – никого пока не удивляет, когда сайт после редизайна почти не напоминает прежний. Так что оптимальным вариантом работы почти всегда является проектирование функциональности сразу на несколько версий вперед.

Итак, определение функциональности. Традиционно требования к функциональности исходят от отдела продаж или от его аналога. Такие требования имеют два источника, а именно жалобы имеющихся клиентов и системы конкурентов. К сожалению, оба эти источника сомнительны. Всегда может оказаться, что желание клиента получить новую функцию обусловлено не реальной потребностью в ней, но собственно концептуальными проблемами системы. Системы же конкурентов страдают как от

такого же непонимания проблемы, так и от множества других причин. Это значит, что прислушиваться к сторонним требованиям к системе, безусловно, следует, но рассматривать эти требования нужно не как директивы, но как признаки общего неблагополучия.

Я в своё время видел техническое задание к системе каталогизации изображений, рассчитанной на домашних пользователей, которая состояла из сведенного в таблицу списка *всех* функциональных возможностей *всех* конкурентов. В результате система получила огромное количество никому не нужных возможностей, что не только никак не помогло ей на рынке, но и безмерно удлинило срок её разработки.

Не копируйте слепо интерфейс конкурентов. Зачем плодить убожество?

Так как всё-таки определить нужную функциональность? Современная наука выдвинула два основных способа, а именно анализ целей и анализ действий пользователей. Эти способы фактически не конфликтуют друг с другом, более того, в процессе определения функциональности желательно использовать оба.

Анализ целей пользователей

Идеей, лежащей в основе данного метода, является простое соображение, гласящее, что людям не нужны инструменты сами по себе, нужны лишь результаты их работы. Никому не нужен молоток, если не нужно забивать гвозди. Никому не нужен текстовый процессор – нужна возможность с удобством писать тексты. Никому не нужна программа обработки изображений – нужны уже обработанные изображения.

Это значит, что сами по себе функции никому не нужны и не важны. Людям нужно средство вообще, делающее возможным выполнять какую-либо работу. Разницу подходов к выбору функциональности в такой системе удобно проиллюстрировать на примере тостера. Стандартный подход, при котором функции выбираются фактически произвольно, в лучшем случае приведет к такому заданию:

> «Нужен ящик с узкой прямоугольной дыркой и нагревателем внутри»

Анализ целей пользователя приведет к другой формулировке:

> «Нужен поджаренный хлеб. Похоже, что проще всего добиться этого созданием ящика с дыркой по форме куска хлеба и нагревателем внутри. С другой стороны, похоже, что этот способ не единственный»

Второй вариант при полном развитии этого метода может привести не только к созданию тостера, но и также и ростера (устройства, в котором можно поджаривать *не только* хлеб).

Главное же другое. Ни в коем случае нельзя дать обмануть себя ненужной конкретикой, т. е. описанием того, *какова* должна быть будущая функциональность. Как правило, одного и того же результата можно добиться несколькими разными способами, при этом важно не только реализовать какой-либо способ, но и выбрать лучший. Анализ целей пользователя как раз и позволяет избежать ненужной конкретики.

Результатом этого процесса должен являться список целей, например, для тостера финальный список целей должен выглядеть очень просто:

> «Должен поджаривать мелкие предметы, преимущественно хлеб»

И только.

После того, как истинные цели пользователей установлены (и доказано, что таких пользователей достаточно много, чтобы оправдать создание системы), приходит время выбирать конкретный способ реализации функции, для чего используется второй метод.

Анализ действий пользователей

Достижение почти всех целей требует от пользователей совершения определенных действий. Разумеется, эти действия могут различаться при разных способах достижения. Например, в «Отелло» одноименный герой Дездемону задушил, что потребовало от него сугубо определенных действий. Если бы он героиню зарезал, ему бы понадобилось действовать иначе. То же самое с отравлением или, например, с выбрасыванием из окна. Все эти действия, разные по сути, привели бы его к одному и тому же результату, а именно к трагическому финалу с последующим раскаянием.

В сколько-нибудь сложных интерактивных системах сами по себе выбранные стратегии действий влияют на требования к функциональности. Возвращаясь к примеру с тостером, можно указать, что помимо возможности поджаривать хлеб он должен включаться и выключаться, более того, он должен быть устроен таким образом, чтобы его можно было удобно мыть. С другой стороны, возможно, что тостер можно сделать так, чтобы он не требовал мойки, в этом случае функция «возможность помыва» становится лишней. Разумеется, взаимодействие человека с тостером очень просто, здесь можно дойти до всего чисто логическим анализом. В компьютерных же системах взаимодействие обычно многократно сложнее, при этом логический анализ неприемлем.

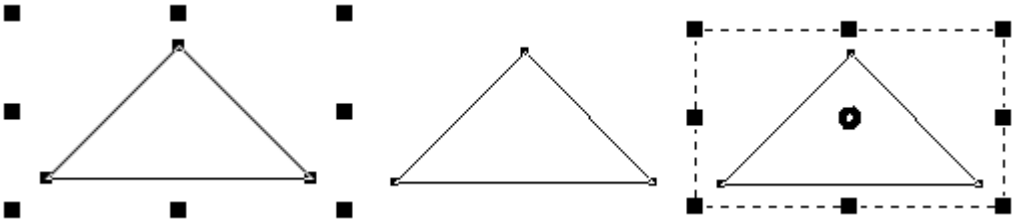


Рис. 61. Пример явной пользы анализа действий пользователей. На рисунке изображены три способа индикации выделенного объекта в векторных иллюстративных программах. Помимо собственно индикации, те же самые элементы управления должны использоваться для изменения формы и размера объектов. Слева элемент управления, введенный в программах фирмы Corel. В программах Corel большие черные прямоугольники позволяют изменять размер выделенного элемента, маленькие же квадратики в углах треугольника (узловые точки), описывающие форму объекта, заблокированы. При выборе инструмента редактирования они разблокируются и появляется возможность изменять форму объекта. В центре и справа – такой же элемент управления программы Macromedia Freehand. По умолчанию изменять размер невозможно, передвигая же узловые точки можно изменять форму объекта. Проблема в том, что изменять размеры объектов нужно гораздо чаще, нежели изменять их форму. Поэтому программы Corel в этом аспекте более производительны. Обнаружить же это можно только путем анализа действий пользователей. Достоинства подхода Corel прекрасно понимают в Macromedia, но поскольку Freehand программа уже устоявшаяся, они не могут просто так скопировать стиль конкурента. Поэтому пришлось ввести новый режим, запускаемый по двойному щелчку на объекте (справа). Что плохо уже тем, что каждый раз при переходе из режима в режим пользователь тратит примерно половину секунды.

Единственным выходом является банальное наблюдение за людьми, выполняющими свою задачу, пользуясь уже имеющимися инструментами, а именно системами конкурентов (если они есть) и предметами реального мира (поскольку очень немного новых действий появилось только после появления компьютеров). Неплохим источником материала для анализа часто служит даже не наблюдение за людьми, но анализ результатов их работы – если оказывается, что результат работы практически не зависит

от используемого инструмента, это значит, что нужна только та функциональность, которая оказала воздействие на результат (т. е. функции, которыми никто не воспользовался, не нужны).

Тут очень важно избежать эгоцентризма. Очень трудно отказаться от мысли «если это нужно мне, это нужно и многим». Возможно, что и нужно. А возможно, что и нет. Единственным же способом проверить, нужна функция или нет, является наблюдение за пользователями и анализ их действий.

Как уже было сказано, обычно есть несколько разных способов реализации функции. Анализ действий пользователей как раз и позволяет определить, какой именно способ следует предпочесть. Поскольку на этом этапе мы узнаём, какая именно функциональность нужна для каждого варианта, можно избрать верный путь по правилу «чем меньше действий требуется от пользователя, тем лучше» (благо компьютер есть, прежде всего, великое средство автоматизации). Не стоит забывать и про другое правило: чем меньше функций, тем легче их сделать.

Низкоуровневые и высокоуровневые функции

Существует два принципиально разных подхода к определению функциональности системы. При первом подходе система снабжается максимальным количеством функций, при этом результаты многих из них являются суммой результатов других функций. При другом подходе все составные функции (метафункции) из системы изымаются. Ярким представителем первого подхода является Corel PhotoPaint, не менее ярким представителем другого – Adobe PhotoShop.

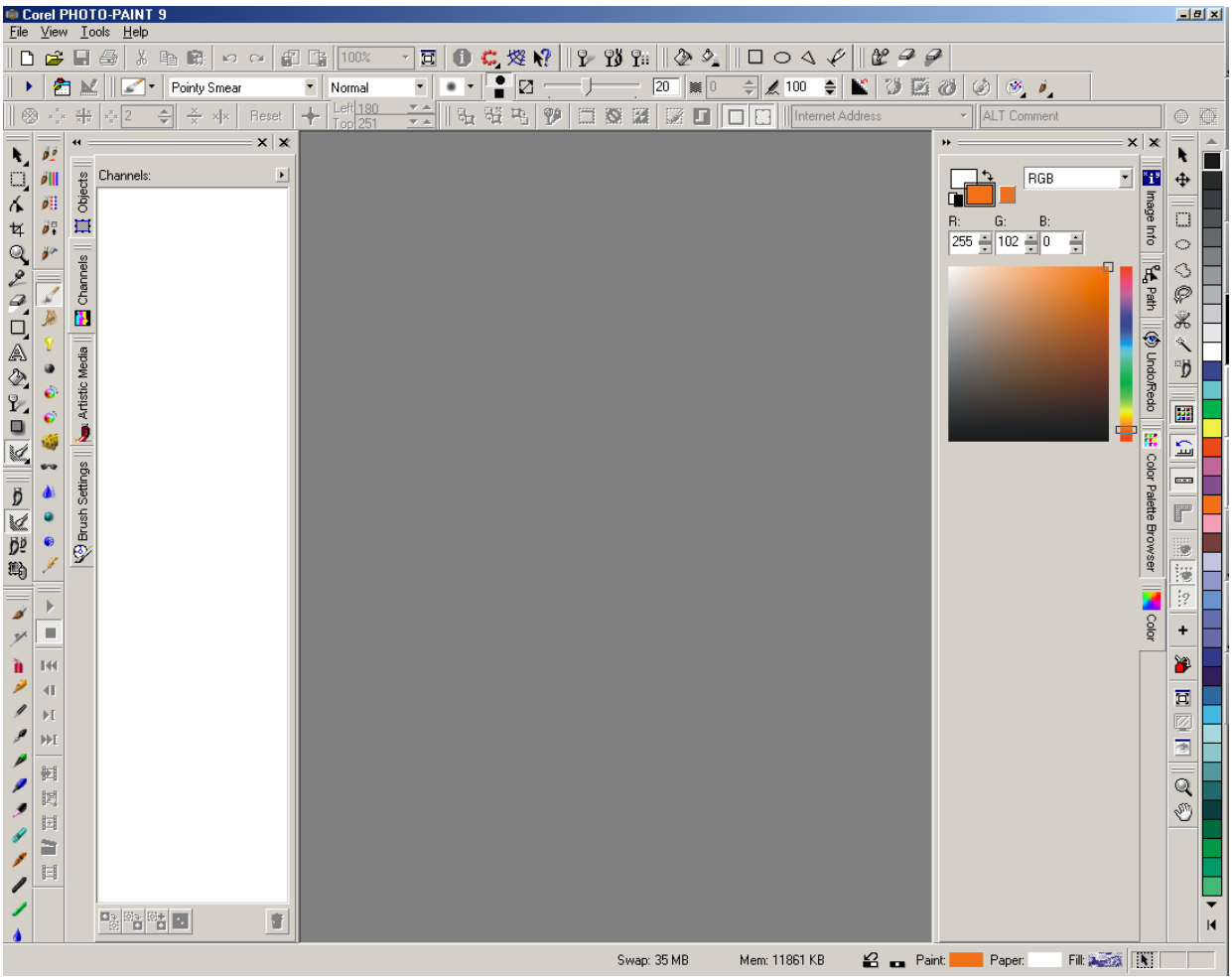


Рис. 62. Окно программы PhotoPaint 9 со всеми интерфейсными элементами, которые можно вывести одновременно (многое не влезло). © Corel.

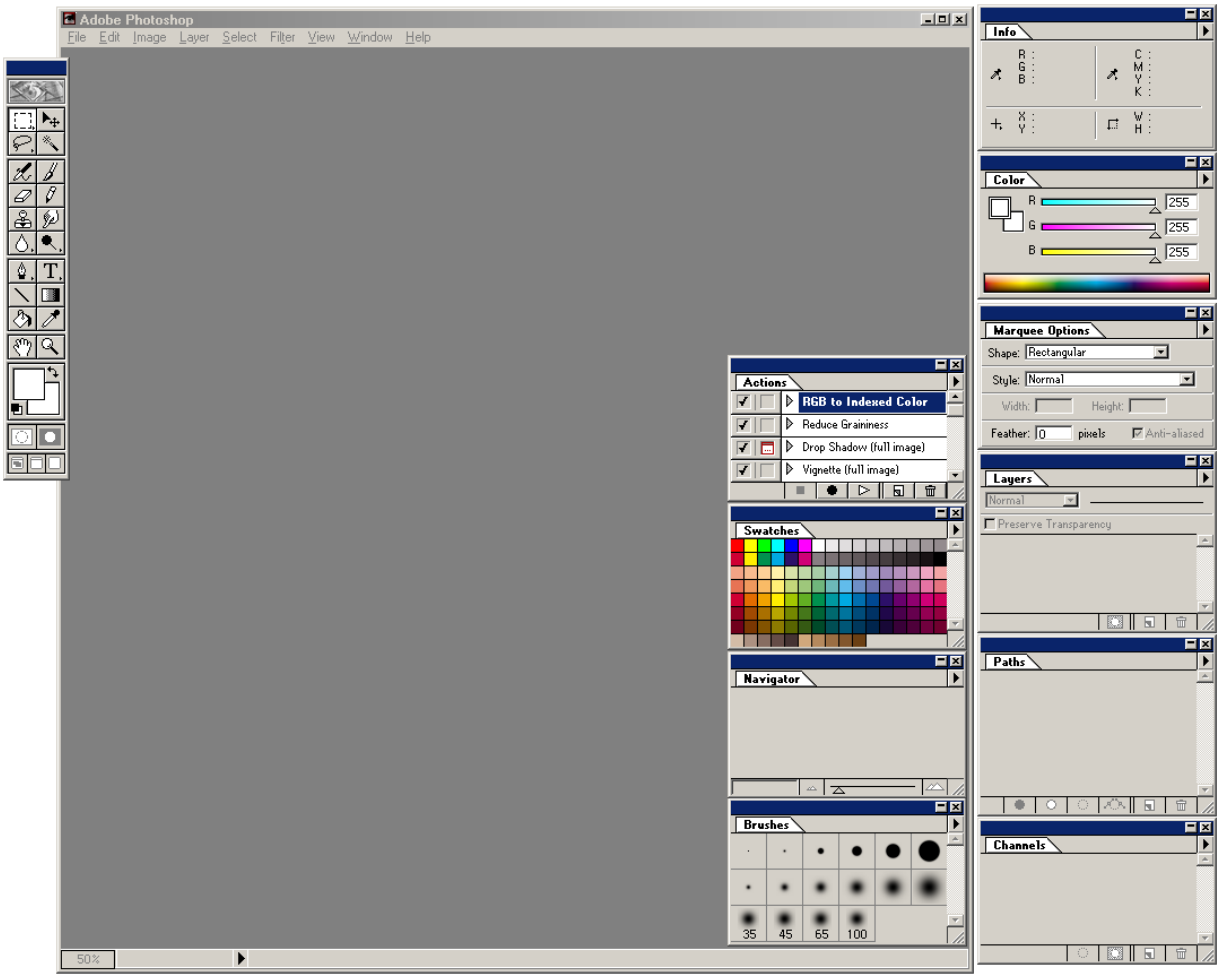


Рис. 63. Окно программы PhotoShop 4, со всеми интерфейсными элементами (влезло всё). В последних версиях PhotoShop появилось несколько новых элементов, так что сравнение не вполне корректно. Тем не менее, в PhotoShop «интерфейса» всё равно гораздо меньше. © Adobe.

Оба подхода имеют как недостатки, так и достоинства. Подход, при котором количество функций ограничено, позволяет упрощать интерфейс, но при этом требует от пользователя понимать, как из многих низкоуровневых функций «собирать» функции более сложные. Подход, при котором помимо низкоуровневых функций есть высокоуровневые, позволяет потенциально обеспечивать большую скорость работы (за счет отсутствия пауз между низкоуровневыми функциями), но зато требует от пользователя знаний о том, где эти высокоуровневые функции найти и как с ними работать, при этом они перегружают интерфейс.

Судя по всему, людям больше нравится пользоваться низкоуровневыми функциями, поскольку это позволяет добиваться более тонких и предсказуемых результатов. С другой стороны, доказательств этого не так уж много (сам факт того, что PhotoShop продается лучше, чем PhotoPaint, говорит не так уж о многом). Таким образом, выбор подхода не слишком прост. С другой стороны, остается возможность компромисса: всегда можно включить в систему средства автоматизации, чтобы пользователи получали возможность создавать (и распространять) свои метафункции, каковой подход как раз и применен в PhotoShop. Я полагаю, что это оптимальный подход.

Создание пользовательских сценариев

Это самый веселый этап работы. Его цель – написать словесное описание взаимодействие пользователя с системой, не конкретизируя, как именно проходит взаимодействие, но уделяя возможно большее внимание всем целям пользователей. Количество сценариев может быть произвольным, главное, что они должны включать все типы задач, стоящих перед системой, и быть сколько-нибудь реалистичными. Сценарии очень удобно различать по именам участвующих в них вымышленных персонажей.

Возвратимся к многострадальному тостеру. Это очень простая система, поэтому для нее достаточно одного сценария.

> «Виктор Х. отрезает кусок хлеба и кладет его в тостер. Он включает тостер и ждет, пока хлеб не поджарится»

Но это просто, ради такого результата не стоит огород городить. Обычно всё несколько сложнее. Предположим, что необходимо разработать сценарии для будущей почтовой программы. Судя по всему (я не анализировал действия пользователей, так что не уверен), для этой задачи необходимо три сценария:

> А) Елизавета Бонифациевна запускает почтовую программу. Она включает процесс скачивания новой почты. Получив почту, она читает все сообщения, затем часть их удаляет, а на одно сообщение отвечает. После чего выключает почтовую программу.
>
> Б) Еремей Карапетович делает активным окно уже открытой почтовой программы и включает процесс скачивания новой почты. Получив почту, он ее читает. Одно сообщение он пересылает другому адресату, после чего удаляет его, а еще одно печатает. После чего переключается на другую задачу.
>
> С) Пришло новое сообщение, и Джамиля Валериевна восприняла соответствующий индикатор. Она делает активным окно почтовой программы и открывает полученное сообщение. Она читает его, после чего перемещает его в другую папку. После чего переключается на другую задачу.

Польза этих сценариев двояка. Во-первых, они будут полезны для последующего тестирования. Во-вторых, сам факт их написания обычно (если не всегда) приводит к лучшему пониманию устройства проектируемой системы, побуждая сразу же оптимизировать будущее взаимодействие. Дело в том, что на таких сценариях очень хорошо заметны ненужные шаги, например, в третьем сценарии гипотетическая Джамиля Валериевна после получения индикатора не смогла сразу же открыть новое сообщение, но должна была открыть окно системы, найти нужное сообщение, открыть его и только тогда прочесть. Понятно, что от этих ненужных этапов смело можно избавиться уже на этой, весьма ранней, стадии проектирования.

Прилив вдохновения

Этот этап либо происходит сам, либо не происходит вовсе. Предыдущие этапы дают достаточное представление о сущности создаваемой системы. Поэтому всегда не вредно несколько задержаться на этом этапе и спросить себя «верно ли то, что методы, которые я собираюсь избрать, самые лучшие? Нельзя ли их сделать принципиально лучше?» Как правило, вдохновение, пришедшее на дальнейших этапах, обходится значительно дороже.

Проектирование общей структуры

Итак, информация о будущей системе собрана. Теперь, пользуясь этой информацией, необходимо создать общую структуру системы («вид с высоты птичьего полета»), т. е. выделить отдельные функциональные блоки и определить, как именно эти блоки связываются между собой. Под отдельным функциональным блоком я понимаю функцию/группу функций, связанных по назначению или области применения в случае программы и группу функций/фрагментов информационного наполнения в случае сайта[1].

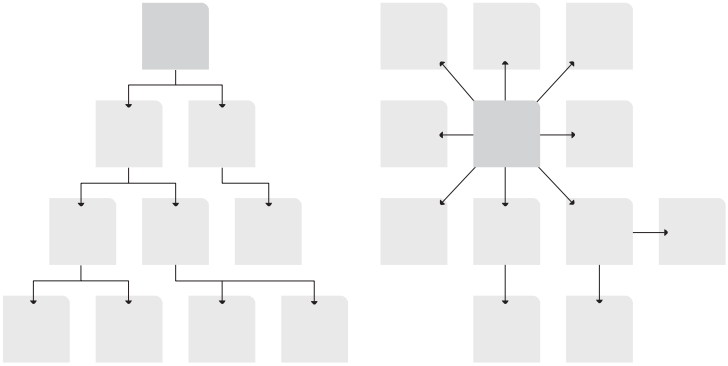


Рис. 64. Типичная структура сайта (слева) и типичная структура программы. Если сайты обычно разветвлены, в том смысле, что функции обычно размещаются в отдельных экранах, то программы обычно имеют только один изменяющийся экран, в котором и вызываются почти все функции. Разумеется, это не догма.

Проектирование общей структуры состоит из двух параллельно происходящих процессов: выделения независимых блоков и определения связи между ними. Если проектируется сайт, в завершении необходимо также создать схему навигации.

Выделение независимых блоков

Для этой работы трудно дать какие-либо конкретные рекомендации, поскольку очень многое зависит от проектируемой системы. Тем не менее, можно с уверенностью рекомендовать избегать помещения в один блок более трех функций, поскольку каждый блок в результирующей системе будет заключен в отдельный экран или группу управляющих элементов. Перегружать же интерфейс опасно.

Результатом этой работы должен быть список блоков с необходимыми пояснениями. В качестве примера рассмотрим гипотетическую программу ввода данных. От пользователя требуется выбрать из списка клиента (или добавить в список нового) и указать, какие именно товары клиент заказал (товары в список тоже можно добавлять). Несколько клиентов постоянно что-то заказывают, так что заставлять пользователя каждый раз искать в списке такого клиента неправильно. При этом блоки разделяются следующим образом:

Основной экран	Навигация между функциями системы
Создание нового заказа	
Добавление существующего товара в заказ	А так же простой поиск товара в списке

1. Я не различаю функции и блоки информационного наполнения из-за того, что сама по себе *функция* сайта – предоставлять информацию. Это значит, что с точки зрения потребителя (не пользователя, а именно потребителя продукта) в интернете нет принципиальной разницы между собственно функциональными возможностями и контентом. Например, поиск по сайту есть явная функция. Но тогда и любое оглавление тоже есть функция сайта – а именно фильтрация информации с определенной целью.

Сложный поиск товара	
Добавление нового товара в список	
Добавление существующего клиента в заказ	А так же простой поиск клиента в списке
Добавление нового клиента в список	
Выбор постоянного клиента	
Обработка заказа	Печать и его переход в папку Исполняемые

Определение смысловой связи между блоками

Существует три основных вида связи между блоками. Это логическая связь, связь по представлению пользователей и процессуальная связь. Логическая связь определяет взаимодействие между фрагментами системы с точки зрения разработчика (суперпользователя). Пользователи имеют свое мнение о системе, и это мнение тоже является важным видом связи. Наконец, процессуальная связь описывает пусть не вполне логичное, но естественное для имеющегося процесса взаимодействие: например, логика напрямую не приказывает людям (с трудом удержался, чтобы не написать «пользователям») сначала знакомиться, а уж потом спать друг с другом, но обычно получается именно так.

Все три типа взаимосвязи должны быть заранее предусмотрены при конструировании системы. Разберем это подробнее.

Логическая связь. С установлением логической связи между модулями обычно проблем не возникает. Важно только помнить, что полученные связи очень существенно влияют на навигацию в пределах системы (особенно, когда система многооконная). Поэтому, чтобы не перегружать интерфейс, стоит избегать как слишком уж отдельных блоков (их трудно найти), так и блоков, связанных с большим количеством других. По моему опыту, для одного блока оптимальным числом связей является число три.

Связь по представлению пользователей. В информационных системах (читай – в интернете), когда необходимо гарантировать, что пользователь найдет всю нужную ему информацию, необходимо устанавливать связи между блоками, основываясь не только на точке зрения разработчика, но основываясь на представлениях пользователей. Дело в том, что чуть ли не единственный распространенный способ поиска, а именно поиск по классификации признаков, работает только в том случае, когда пользователи согласны с принципами этой классификации. Большинство же понятий однозначно классифицированы быть не могут из-за наличия слишком большого количества значимых признаков. Также проблема состоит в том, что реальный классификационный признак может отличаться от широко распространенного. Например, нужно как-то классифицировать съедобные растения. Помидор, который почти все считают овощем, на самом деле ягода. Не менее тяжело признать ягодой арбуз. Это значит, что классификация, приемлемая для ботаника, не будет работать для всех остальных, причем обратное не менее справедливо.

В то же время, существует очень простой способ создания классификации, с явной ненаучностью сочетающий не менее явную практическую пользу. Способ этот называется карточной сортировкой, при этом его название полностью совпадает с его сущностью. Все понятия, которые требуется классифицировать, пишутся на бумажных карточках из расчета «одно понятие – одна карточка». После этого группе пользователей из целевой аудитории предлагается эти карточки рассортировать (при этом каждый субъект получает свой набор карточек). Получившиеся кучки из карточек нужно разобрать на составляющие и свести результаты от разных субъектов в один способ классификации.

Ничего более работоспособного до сих пор человечеством не придумано. В то же время этот способ имеет определенные недостатки: во-первых, трудно заполучить на несколько часов представителей целевой аудитории, а во-вторых, при малом количестве субъектов результаты могут быть сомнительны (как минимум, нужно 4-5 человек).

Процессуальная связь. Установление качественной процессуальной связи обычно довольно трудная задача, поскольку единственным источником информации является наблюдение за пользователями. В то же время установление такой связи дело исключительно полезное. Зачем, например, рисовать на экране сложную систему навигации, если точно известно, к какому блоку пользователь перейдет дальше? В этом смысле зачастую оправдано навязывать пользователю какую-либо процессуальную связь, жертвуя удобством, зато выигрывая в скорости обучения (поскольку пользователю приходится думать меньше). Жестко заданная связь позволяет также уменьшить количество ошибок, поскольку от пользователя при ней не требуется спрашивать себя «не забыл ли я чего?». Хорошим примером жестко заданной процессуальной связи является устройство мастеров (wizards, см. «Последовательные окна» на стр. 98), при котором пользователя заставляют нажимать кнопку Далее.

В конце этого этапа должна получиться примерно такая схема (в примере используется та же задача с вводом данных):

Результат

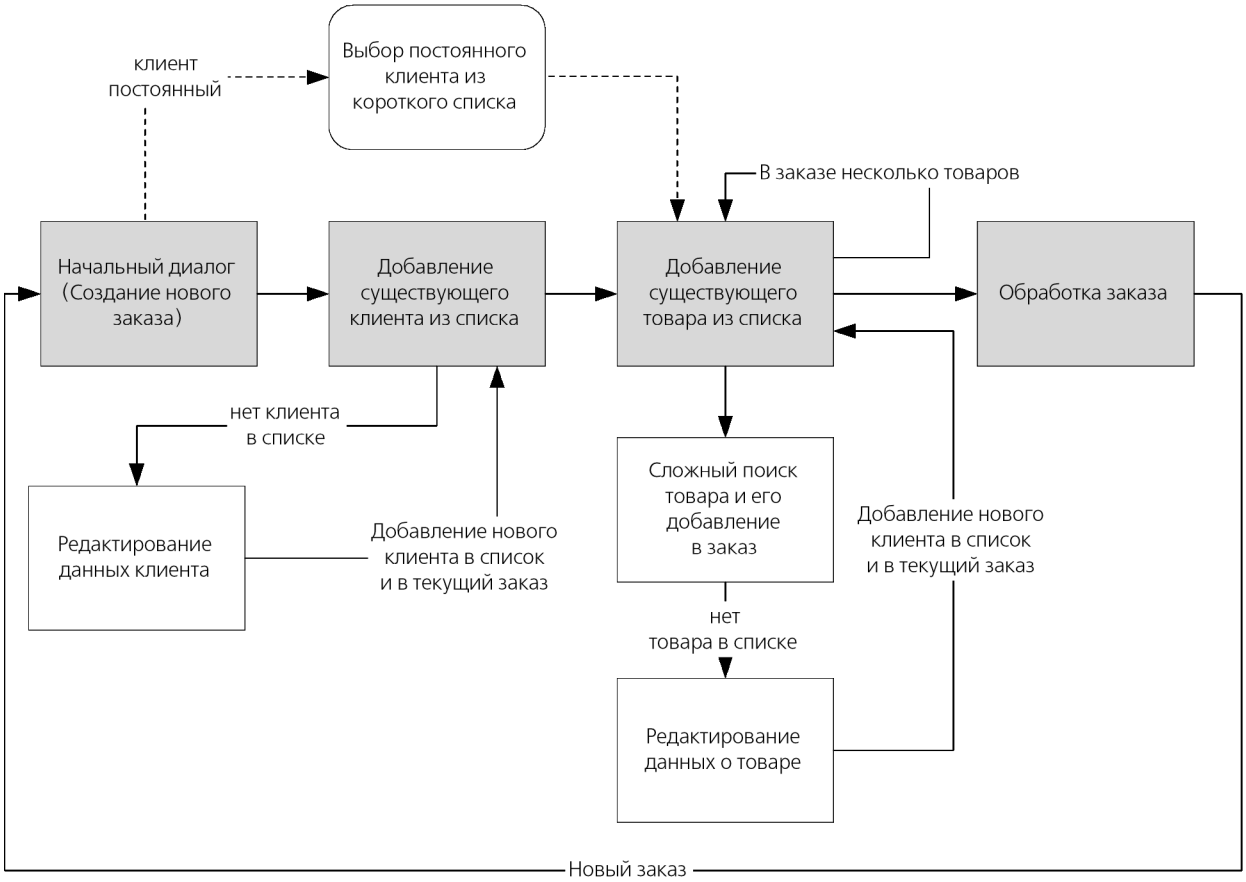


Рис. 65. Пример общей схемы (вторая версия). Прямоугольник обозначает отдельный экран, прямоугольник со скругленными углами – область экрана, пунктирная линия – альтернативное действие. Обратите внимание, что в этой схеме интерфейс заставляет пользователя выполнять задачу в сугубо определенной последовательности.

Существует любопытная закономерность: чем эстетически привлекательней выглядит схема (без учета цветового кодирования и веселеньких шрифтов), тем она эффективней. Всегда надо стараться сделать схему возможно

более стройной и ясной. Например, показанная схема изначально выглядела значительно хуже. Я потратил некоторое время на её доводку, при этом, разумеется, нашел пару ошибок.

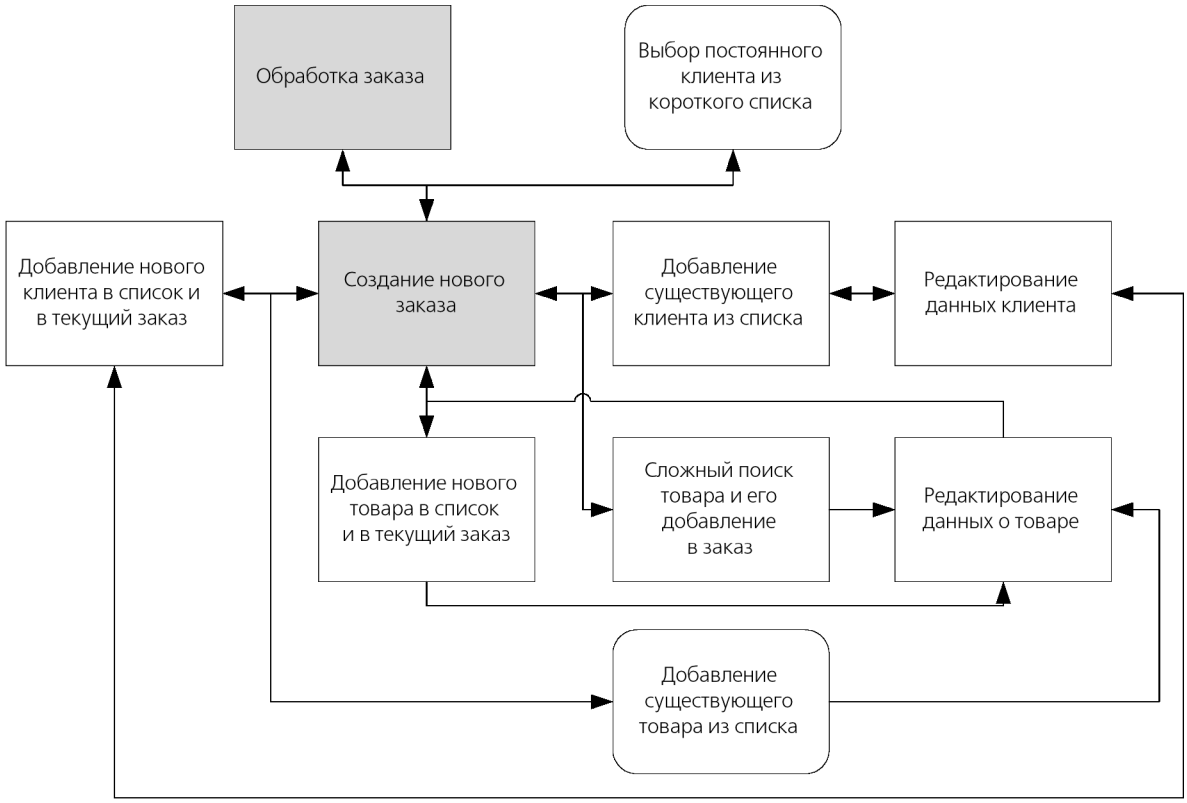


Рис. 66. Первая версия схемы. Свободное взаимодействие человека и системы (пользователь волен выполнять работу в произвольной последовательности).

Рисовать такие схемы очень удобно в MS Visio или подобной ей системе. Результат, в особенности сложно выглядящий и напечатанный на листе бумаги большого формата, очень хорошо смотрится на стене и животворяще действует на заказчиков и руководство (далее вы узнаете, как сделать эту схему еще более красивой и непонятной).

Проектирование отдельных блоков

Итак, теперь вы знаете, сколько экранов (страниц) вам нужно и что должно происходить на каждом экране. Настало время проектировать отдельные экраны.

Это, пожалуй, самая сложная часть работы (не считая наблюдения за пользователями). Хуже того, она плохо поддается алгоритмизации. Именно на этом этапе, помимо пачки бумаги с карандашом, пригодится вам содержимое первых двух частей этой книги. Но помимо этого есть еще две вещи, которые вам нужно узнать: GOMS и адаптивная функциональность.

Предсказание скорости

Часто приходится выбирать между разными вариантами реализации интерфейса, причем отбрасывать варианты жалко, потому что они хорошие. Можно, конечно, сделать несколько прототипов и протестировать их на пользователях, но это довольно длительный и трудоемкий процесс. К счастью, есть метод оценки интерфейса, позволяющий быстро выбрать лучший вариант.

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)[1].

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 Keystroke-level Model (KLM). К сожалению, этот вариант метода имеет определенные недостатки (что, впрочем, уравновешивается его простотой):

- 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
  Разумеется, согласно закону Фитса (см. «Быстрый или точный» на стр. 10), время, затрачиваемое на перемещение курсора, зависит как от дистанции, так и тот размера цели. Тем не менее, это число представляет достаточно точный компромисс.
- Взятие или бросание мыши (В): 0,4 сек
- Продолжительность выбора действия (Д): 1,2 сек
  В среднем, за 1.2 секунды пользователь принимает решение, какое именно действие он должен совершить на следующем шаге. Обычно это самый сложный оператор, поскольку часто непонятно, в каких именно местах процедуры его необходимо ставить. Например, иногда, когда пользователь совершал искомую последовательность действий не раз и при этом совершенно уверен в том, что общий ход процедуры не будет отличаться от обычного, это время затрачивается только в самом начале выполнения (далее действия будут совершаться автоматически). С другой стороны, начинающим пользователям приходится выбирать действие перед каждым своим шагом. Однако в большинстве случаев достаточно считать, что это время нужно добавлять перед всеми нажатиями, которые не приходятся на область с установленным фокусом, перед всеми командами, инициированными мышью и после существенных изменений изображения на экране (но и здравый смысл тут не помешает). С практической точки зрения важнее устанавливать этот оператор везде одинаково, нежели устанавливать его возможно более точно.
- 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.

1. Card, Moran, and Newell, «The Psychology of Human-Computer Interaction», Erlbaum 1983. См. также David Kieras, «Using the Keystroke-Level Model to Estimate Execution Times», 1993 (доступно в интернете).

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:

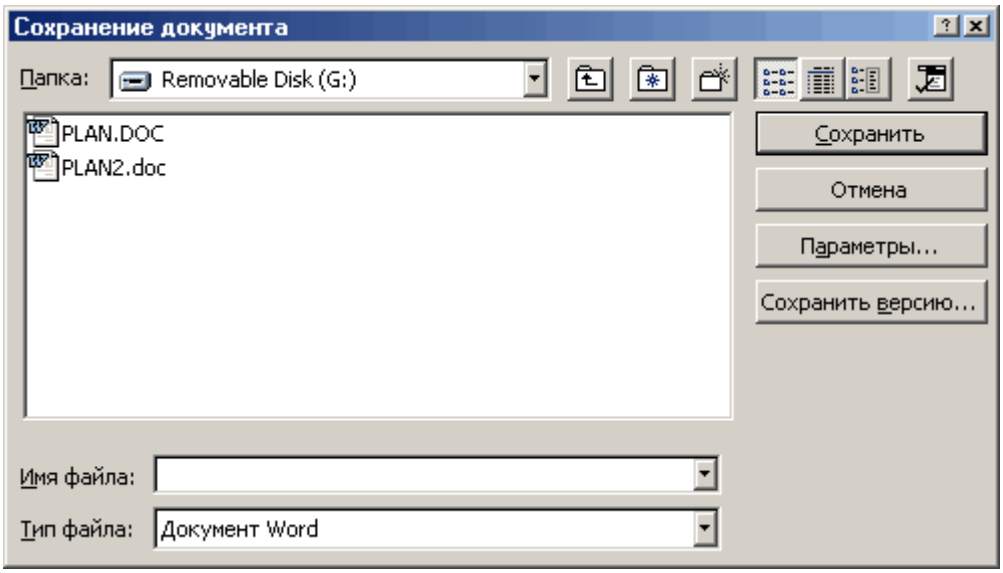


Рис. 67. Диалоговое окно сохранения файла в Word. © Microsoft.

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

Тип действия	Продолжительность	Комментарий
К x 6	0,28 x 6	Ввод названия файла. В слове Опись пять букв, при этом к ним добавляется нажатие на кнопку Shift (поскольку первая буква прописная).
В	0.4	Перенос руки с клавиатуры на мышь
П	1,1	Перемещение курсора к кнопке Сохранить
М	0,1	Нажатие кнопки мыши
Р	0,1	Время реакции системы – диалоговое окно сохранения файла закрывается, а файл сохраняется
Д	1,2	Пользователь вспоминает, как закрывается программа
П	1,1	Перемещение курсора к меню Файл
М	0,1	Нажатие кнопки мыши
Д	1,2	Открылось меню, и пользователю необходимо найти элемент Выход
М	0,1	Нажатие кнопки мыши
Р	0,1	Время реакции системы – программа закрывается

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 x 7	0,28 x 7	Ввод названия файла. В придачу к шести изначальным нажатиям, пользователь нажимает клавишу Enter, сразу инициируя запись файла

Тип действия	Продолжительность	Комментарий
P	0,1	Время реакции системы – диалоговое окно сохранения файла закрывается, файл сохраняется, после чего закрывается и программа

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.

Адаптивная функциональность

Помимо общей логики работы, в системе должна быть ещё одна логика, упрощающая первую и делающую работу пользователя более простой и естественной. Я называю эту вторую логику адаптивной функциональностью.

Возьмем пульт от телевизора. Телевизор выключается только одной кнопкой на пульте, но включается от нажатия любой кнопки. Это не следует напрямую из логики системы, но это естественно. Когда на этаж приезжает лифт с неавтоматическими дверями, дверь можно открыть ещё до того, как погаснет кнопка вызова (чтобы лифт не увели). Это не вполне логично, но естественно. Другой известный, но не всеми осознаваемый, пример: когда Windows при входе в систему спрашивает пароль, нужно нажать Ctrl+Alt+Delete. В этом же диалоговом окне есть кнопка Справка, нажатие на которую открывает ещё одно диалоговое окно, повествующее о том, как нажать эти три клавиши. Так вот, чтобы войти в систему, это окно не нужно закрывать, нажать Ctrl+Alt+Delete можно по-прежнему. С системной точки зрения это неправильно (почему пользователь не закрыл сначала окно с подсказкой?), но для пользователей это естественно.

Все три примера демонстрируют готовность системы (а точнее, её разработчиков) усложнить свою логику, чтобы упростить логику пользователя. Результат: систему легче использовать. Я вообще полагаю, что наличие адаптивной функциональности служит отличным индикатором качества дизайна системы. Систему, которая не подстраивается под пользователей, невозможно назвать зрелой.

Остается один вопрос: как определить, какие фрагменты и функции системы должны быть адаптивными? Ответ: единственным решением является детальный анализ взаимодействия пользователей с системой. Помочь здесь может только тестирование интерфейса на пользователях.

Результат

Итак, у вас есть куча мятых бумажек, на которых нарисованы все диалоговые окна. Как можно скорее перерисуйте их на компьютере (чтобы не растерять). При этом вы найдете несколько не замеченных ранее ошибок. Исправьте их.

Создание глоссария

Еще в процессе проектирования полезно зафиксировать все используемые в системе понятия. Для этого нужно просмотреть все созданные экраны и выписать из них все уникальные понятия (например, текст с кнопок, названия элементов меню и окон, названия режимов и т.д.). После этого к получившемуся списку нужно добавить определения всех концепций системы (например, книга или изображение).

Теперь этот список нужно улучшить. Для этого:

- Уменьшите длину всех получившихся элементов.
- Покажите этот список любому потенциальному пользователю системы и спросите его, как он понимает каждый элемент. Если текст какого-то элемента воспринимается неправильно, его нужно заменить[1].
- Уменьшите длину всех получившихся элементов.
- Проверьте, что одно и то же понятие не называется в разных местах по-разному.
- Уменьшите длину всех получившихся элементов.
- Проверьте текст на совпадение стиля с официальным для выбранной платформы (если вы делаете программу, эталоном является текст из MS Windows).
- Уменьшите длину всех получившихся элементов.
- Убедитесь, что на всех командных кнопках стоят глаголы-инфинитивы (Задушить, Отравить, Выкинуть из окна).

После чего список нужно повесить на стену и стараться не менять его в будущем.

Сбор полной схемы

К этому моменту вы обладаете:

- общей схемой системы
- планами отдельных экранов
- глоссарием.

Пора свести всё это воедино. Работа эта довольно скучная и утомительная, но и от неё есть существенная польза. Во-первых, рисовать такую схему гораздо легче, чем делать прототип, множество же ошибок можно выловить и в ней, не переделывая прототипа. Во-вторых, прототип после окончания проекта вы выкинете, а схему можно повесить на стену и скромно говорить про нее, что, дескать, работа простенькая, но тоже ничего себе. Посетители будут смотреть на непонятную, в половину стены, схему с уважением, а на вас – с обожанием и суеверным ужасом. Врачам, например, для такого отношения к себе приходится резать трупы; по сравнению с этим рисование схемы кажется пустяком.

Рисовать схему очень удобно в уже упоминавшейся Visio. Результатом должен выглядеть примерно так (разумеется, рисовать дублирующиеся куски системы необязательно):

1. Результат будет особенно эффективен, если получится создавать глоссарий, работая в тесном контакте с целевыми пользователями. Неоднократно было замечено, что обычные пользователи придумывают значительно более работоспособные названия элементов, нежели эксперты, например: B. Fischhoff, D. McGregor, L. Blackshaw. Creating Categories for Databases. International Journal of Man-Machine Studies, 33-63 1987, 27 pp.

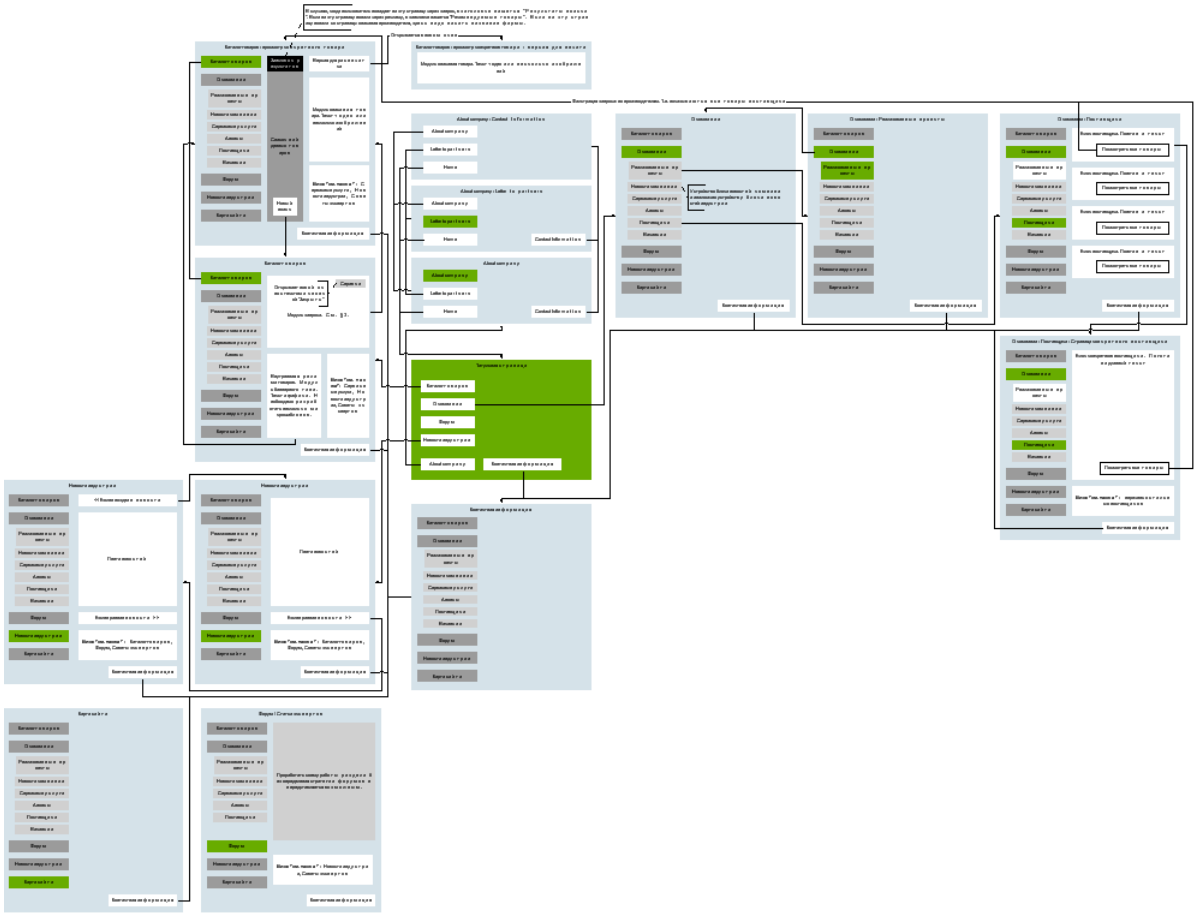


Рис. 68. Пример готовой схемы интерфейса сайта.

Если серьезно, то самой важной целью этого этапа является создание плана обработки системой исключительных ситуаций интерфейса. Необходимо определить, что делать системе, если пользователь вызвал команду, которую для этого конкретного объекта выполнить невозможно (например, пользователь пытается послать письмо человеку, почтовый адрес которого системе неизвестен).

Проверка схемы по сценарию

Последней задачей перед построением прототипа является проверка внутренней логики системы. Дело в том, что всегда существует вероятность того, что вы где-то что-то забыли или спланировали неправильно. Как уже было сказано, исправить эти ошибки лучше всего до построения прототипа (даже первой его версии). Конечно, многие структурные ошибки нельзя найти никакими методами, кроме длительного логического анализа. С другой стороны, практика показывает, что почти все найденные ошибки будут существенными. Так что лишняя проверка не повредит.

Для финальной проверки схемы вам пригодятся разработанные вами пользовательские сценарии. Не глядя на схему, необходимо подробно описать, как все вымышленные пользователи будут взаимодействовать с системой, не пропуская ни одного элемента управления. После чего сверить полученный текст со схемой.

Тут возможно три варианта развития событий: либо вы обнаружите, что что-то забыли задокументировать в схеме, либо обнаружите, что свежеописанный рассказ значительно лучше схемы, вероятнее же всего, что и то и другое произойдет одновременно. На четвертый вариант, а именно на полное отсутствие проблем, рассчитывать, как правило, не стоит.

Экспертная оценка

Весьма эффективным средством оценки получающегося интерфейса является его экспертная оценка. Часто оказывается, что сравнительно дорогое тестирование показывает то, что было бы легко видно постороннему, тем более вооруженному опытом и квалификацией, взгляду. Хотя экспертная оценка не может быть полноценной заменой тестирования, она обладает одним существенным преимуществом – для её проведения не требуется прототип. Это значит, что эксперт может быть приглашен на ранних стадиях работы, когда польза от обнаружения ошибок максимальна.

Для проведения экспертной оценки нужно знать следующее:

- Разные люди обнаруживают разные ошибки. Это значит, что метод работает лучше, когда количество экспертов больше единицы.
- Лучше привлекать несколько экспертов не одновременно, но последовательно.
- Чем больше информации о проектируемой системе будет предоставлено эксперту, тем более сложные проблемы он сможет выявить.
- Нельзя требовать от эксперта работы по весу. В большинстве случаев результатом его работы будут одна или две страницы текста (поскольку описание одной проблемы требует обычно всего двух или трех предложений). Если от эксперта будет требоваться объемный результат работы, он включит в него много несущественных подробностей.
- Координаты всех людей, способных выполнить такую работу, можно найти на сайте www.usability.ru.

Построение прототипа

Итак, первый этап пройден. У вас есть полная схема, описывающая всё взаимодействие пользователя с системой. Настало время делать прототип системы для тестирования.

При создании прототипа наиболее частой ошибкой является чрезмерное наведение глянца и вообще стремление сделать прототип возможно более похожим на результирующую систему. В самом таком подходе нет ничего плохого (всё равно определенные части прототипа приходится делать максимально совершенными), проблема в том, что в большинстве случаев прототип после тестирования оказывается неправильным. Его приходится переделывать, причем иногда полностью, при этом все инвестированные в прототип ресурсы оказываются выброшенными на ветер.

Не полируйте прототип

Поэтому всегда правильно делать прототип настолько похожим на результирующую систему, насколько версия прототипа поздняя. Первый прототип стоит делать максимально примитивным. Только после того, как тестирование подтверждает его правильность, стоит делать более детализированный прототип.

Итак, как быстрее и дешевле построить прототип?

Первая версия. Бумажная

Необходимо нарисовать на бумаге все экраны и диалоговые окна (читай – распечатать соответствующие части схемы). Нужно только убедиться, что все интерфейсные элементы выглядят единообразно и сколько-нибудь похоже на реальные. Эта распечатка и является первым прототипом. На нём вполне можно тестировать восприятие системы пользователем и её основную логику.

Польза начального прототипирования на бумаге заключается, во-первых, в исключительной простоте модификации по результатам тестирования, а во-вторых, в возможности безболезненно отлавливать представителей целевой аудитории. Значительно легче лестью и коварством завлечь субъекта к письменному столу, нежели к компьютеру, на котором надо что-то запускать, ждать, пока запустится и так далее.

Разумеется, значение слова «версия», употребляемого в этой главе, весьма условно. В действительности после обнаружения каждой ошибки схема и прототип исправляются, а тестирование продолжается уже на новом прототипе. Так что на этом этапе прототип может пережить множество исправлений и, соответственно, много версий.

Вторая версия. Презентация

После исчерпания возможностей бумажной версии прототипа стоит создать новую версию (исправив, разумеется, уже обнаруженные проблемы). Для этого точно так же рисуется интерфейс, но уже не на бумаге, но в какой-либо презентационной программе (MS PowerPoint, например). При этом каждый экран получает отдельный слайд, а результат нажатия кнопок имитируется переходами между слайдами (благо во всех презентационных программах есть возможность установки гиперссылок)[1].

С этой версией прототипа можно тестировать значительно более сложное взаимодействие человека с системой, нежели с бумажной. С другой стороны, исправление найденных ошибок значительно более трудоемко.

Фактически для большинства систем этой версии оказывается достаточно.

Третья версия

В тех случаях, когда в интерфейсе появляются нестандартные элементы или необходимо проверить реальную скорость взаимодействия пользователя с системой, создается еще одна версия прототипа – реально выглядящая, но лишенная каких-либо алгоритмов и, соответственно, не показывающая реальных данных. Делать этот вариант можно как в средах разработки, благо в них есть визуальные инструменты создания интерфейсов, так и в редакторах изображений, что обычно быстрее. Фактически при этом создаются фальшивые снимки экрана, на которых и производят тестирование. Понятно, что существенно модифицировать эти экраны затруднительно, так что лучше не увлекаться такой работой, не получив каких-либо гарантий ее правильности.

1. В последних версиях MS Visio появилась стабильно работающая функциональность, позволяющая забыть о PowerPoint: в Visio всё получается лучше.

Иногда необходимо тестировать взаимодействие пользователя не только с интерфейсом системы, но и с обрабатываемыми системой данными. Например, работая с графической программой, пользователь не только нажимает на экранные кнопки, но также создает и модифицирует изображения мышью. Область же редактирования данных зачастую вообще не содержит каких-либо визуальных интерфейсных элементов, из чего вовсе не следует, что интерфейса в ней нет, его, наоборот, много. Другой разговор, что счет в нем идет не на кнопки и переключатели, но на пиксели и миллисекунды.

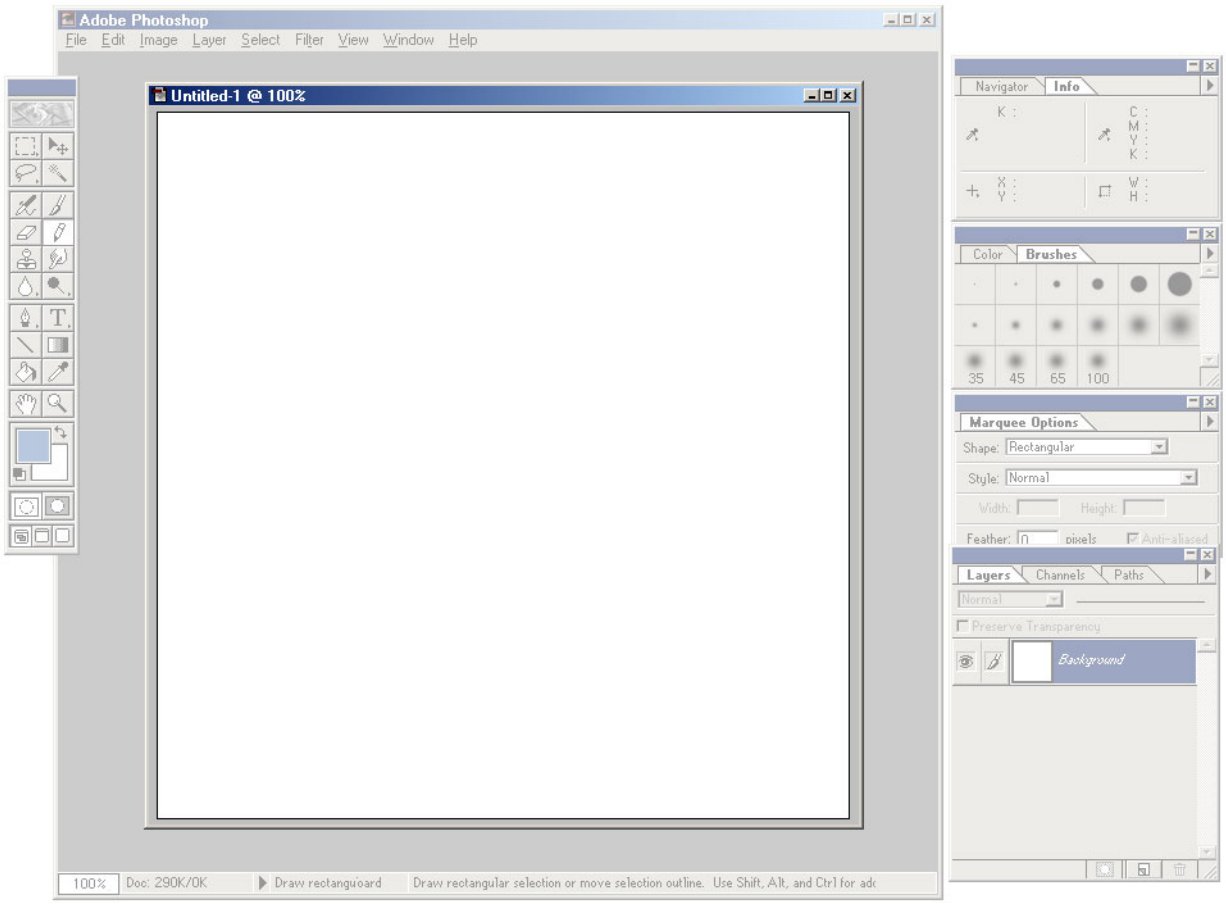


Рис. 69. В окне Adobe PhotoShop количество интерфейса весьма значительно. В то же время наиболее важное для пользователя действие, а именно изменение изображения, создается в области вообще без интерфейсных элементов. © Adobe.

Понятно, что создание прототипа в таких условиях не поможет, поскольку прототип вообще не будет отличаться от проектируемой системы. В таких условиях лучше всего убедить программистов написать нужные участки кода до написания всего остального, и проводить тестирование уже на реальной системе.

Тестирование/модификация прототипа

Какими бы не были совершенными логические соображения, приведшие к созданию интерфейса, всегда остается вероятность того, что интерфейс получился плохой, либо, что более вероятно, не такой хороший, каким бы он мог быть. Необходимо иметь какие-либо подтверждения его работоспособности. К счастью, проверка качества интерфейса обычно непроблематична. Всё, что для этого нужно, это несколько пользователей средней квалификации, никогда не видевшие тестируемой системы, плюс прототип (разумеется, при наличии основательного бюджета можно развернуться и пошире, например, купить прибор, фиксирующий направление взгляда пользователя).

В литературе часто встречается мнение, что тестированием можно решить чуть ли не все проблемы интерфейса. Утверждение это сомнительно. Тестированием, скорее, можно определить слабые места интерфейса, но почти невозможно обнаружить сильные, поскольку они пользователями просто не замечаются, и совсем уж невозможно определить новые способы улучшения. Происходит это из-за того, что субъекты тестирования:

- не обладают всей необходимой информацией о системе,
- ничего не знают о проектировании интерфейсов,
- их мотивация существенно отличается от необходимой – вместо того, чтобы стремиться сделать хороший интерфейс, они стремятся оставить в этом интерфейсе свой след («Вася здесь был»).

Вообще, слушать потребителей обычно неправильно. Разве мы спрашиваем канарейку, в какой клетке она хочет жить? Сюжет про американский автопром, например, стал уже частью истории бизнеса: все американские потребители в семидесятых годах дружно утверждали, что они хотят большие, мощные машины, при этом так же дружно покупая маленькие и маломощные японские автомобили. Или другой пример – в советское время измученные коммунизмом люди мечтали вовсе не об отпуске на Тенерифе, о котором они ничего не знали, но о финском хромированном смесителе, который поставил себе сосед – хотя Тенериф, безусловно, в качестве мечты интереснее.

В то же время, даже не слушая пользователей, обязательно нужно принимать во внимание их потребности, способности и предпочтения. Например, нередко дизайнер интерфейса знает о предметной области меньше, нежели будущие пользователи. В таких условиях потеря контакта с пользователями грозит крахом продукта, просто потому, что система оказывается неспособна решать задачи, о которых дизайнер ничего не знал. Чаще, однако, случается так, что уровень «компьютерной грамотности» дизайнера оказывается выше уровня аудитории. В этом случае все получается ещё хуже: дизайнер выбирает решения, которые обеспечивают эффективность работы, а потребителям нужны решения, которые они могут понять, в результате они оказываются неспособными воспользоваться системой (это совершенно нормальная ситуация, особенно в интернете, где ROI (см. «Почему пользователи учатся» на стр. 22) необыкновенно низок). Например, замечено, что опытные пользователи (к которым относятся дизайнеры) создают значительно менее

работоспособные иерархии меню, нежели пользователи начинающие. Разумеется, если переоценка способностей реальных пользователей и незнание предметной области совпадают, результат бывает ещё хуже.

Постановка задачи

Одной из самых важных предпосылок успешного тестирования является правильная постановка задачи. Всегда есть шансы потратить несколько часов в поисках ответа на ненужный вопрос. Хуже того – случается, что после окончания длительного и утомительного сеанса приходит понимание того, что тех же результатов можно было бы добиться с меньшими трудозатратами. Правильная постановка задачи позволяет этих проблем избежать.

Иногда имеющийся вопрос можно переформулировать таким образом, чтобы он сам по себе вел к ответу. Почти всегда – чтобы метод ответа на него обходился дешевле. В любом случае невредно сначала убедиться в том, что ответ на задаваемый вопрос действительно нужен.

Например, нужно определить, как пользователи видят какое-либо диалоговое окно. Можно нацепить на тестера уже упомянутый прибор для определения направления взгляда, а потом долго определять, куда тестер смотрел. Можно найти неопытного пользователя, который помогает себе мышью (многие пользователи постоянно перемещают курсор мыши в том место, куда они смотрят). А можно переформулировать вопрос и поступить совсем иначе (но об этом позже).

Собственно тестирование

Технически сеанс тестирования довольно прост. Нужно иметь несколько пользователей, которые ни разу не видели текущего состояния системы. За исключением редких случаев, когда ваша система рассчитана на продвинутых пользователей (power user), нужно подбирать не слишком опытных субъектов. Тестерам дается задание из ранее написанных сценариев, они его выполняют, после чего результаты анализируются. Идея проста, тем не менее, по этому поводу написано довольно много литературы (причем объемной). Впрочем, в большинстве случаев достаточно помнить следующее:

- Тестирование на одном пользователе позволяет найти примерно 60% ошибок. Соответственно решайте сами, сколько пользователей необходимо для одного сеанса.
- Если у вас есть возможность оставить тестера одного, не пренебрегайте этим. Одностороннее зеркало в таких условиях не роскошь.
- Никогда не прерывайте пользователя. Никогда не извиняйтесь за несовершенство тестируемой системы. Никогда не говорите «Мы потом это исправим». Никого не обвиняйте. Никогда не называйте процесс тестирования «пользовательским тестированием» – пользователь решит, что тестируют его, и будет бояться.

Проверка посредством наблюдения за пользователем

Один из самых простых видов тестирования. Пользователю дается задание, он его выполняет, его действия фиксируются для дальнейшего анализа какой-либо программой записи состояния экрана (удобен Lotus ScreenCam). Чтобы пользователь не тревожился и не стеснялся, его лучше всего оставить в одиночестве.

Метод исключительно полезен для выявления неоднозначности элементов интерфейса. Поскольку каждая неоднозначность приводит к пользовательской ошибке, а каждая такая ошибка фиксируется, обнаружить их при просмотре записанного материала очень легко.

Этот тест замечательно подходит для поиска проблем интерфейса. Кроме того, если замерять время выполнения задания (секундомером), можно оценить производительность работы пользователей. Этот же метод позволяет посчитать количество человеческих ошибок.

Мыслим вслух

Метод довольно нестабильный, но порой дающий интересные результаты (очень зависит от разговорчивости пользователя-испытателя). Соответствует проверке посредством наблюдения за пользователем, но тестера при этом просят также устно комментировать свои действия. Затем комментарии анализируются.

Метод позволяет легко определить недостатки реализации конкретных интерфейсных идей (неудачно расположение элементов, плохая навигация). Обратите внимание, что субъект, проговаривающий свои впечатления, работает медленней обычного, так что измерять скорость работы этим методом невозможно.

Проверка качества восприятия

Тест позволяет определить, насколько легко интерфейсу обучиться. Поскольку существует разница между понятиями *видеть* и *смотреть*, а запоминается только то, что увидено, необходимо обладать уверенностью в том, что пользователь увидит если не всё, то уж хотя бы всё необходимое. А значит – запомнит, благодаря чему в будущем ему не придется обшаривать меню в поисках «чего-то такого, что, я точно знаю, где-то здесь есть».

Сама по себе методика проста. Пользователю даётся задание, связанное с каким-либо отдельным диалоговым окном. Пользователь его выполняет. Через несколько минут пользователя просят нарисовать (пускай даже грубо и некрасиво) только что виденное им окно. После чего рисунок сравнивается с оригиналом.

Разумеется, пользователь запоминает только то, что ему кажется актуальным в процессе работы с окном (плюс еще что-нибудь за того, что ему показалось интересным, да и то не всегда). Это один из тех редких случаев, когда срабатывает ограничение на объем кратковременной памяти, так что количество запомнившихся элементов управления не может быть выше порога. Например, пользователь, которому нужно сменить шрифт абзаца на Arial, из всего окна выбора шрифта в MS Word запоминает только три элемента управления (разумеется, он помнит, что помимо них были и другие, но точно вспомнить остальные элементы он, как правило, не может).

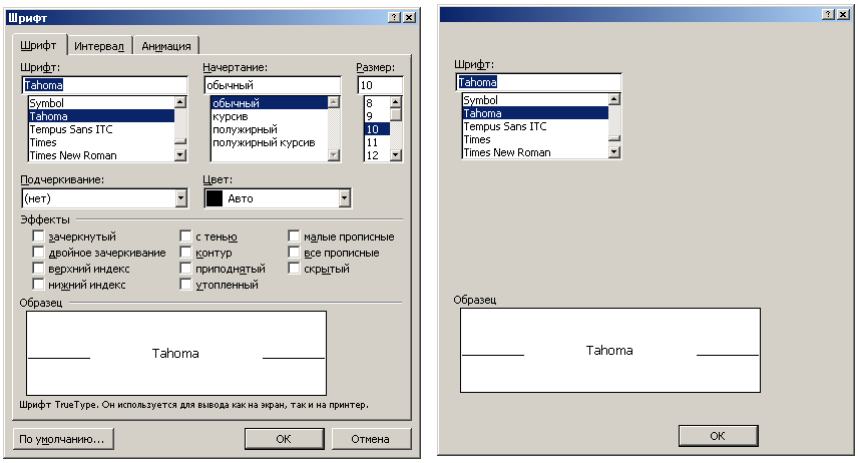


Рис. 70. Диалоговое окно (слева) и то, что из него запоминается. В зависимости задачи, пользователь использует различные элементы интерфейса, при этом запоминаются только используемые элементы. © Microsoft.

Как это ни грустно, основное предназначение этого теста состоит в том, чтобы раз за разом убеждаться в том, что запомнить нужное совершенно невозможно. Но и в таком качестве он полезен.

А потом – всё переделать!

Тестирование само по себе имеет существенный недостаток: если тестирование проблем не выявило, получается, что оно было проведено зря. Если выявило, придется проблемы решать, что тоже существенная работа. Таким образом, сама идея тестирования интерфейса создает конфликт интересов у дизайнера – работы от него прибавляется либо много, либо очень много – но всегда прибавляется. Работать же, разумеется, не хочется.

Именно поэтому тестирование бессознательно переносят на самое окончание проекта, когда что-либо исправлять уже поздно. В результате тестирование показывает, что проект сделан плохо, что никому не нравится, включая его создателя, после чего результаты проверки прячутся в дальний ящик.

В то время как сама по себе идея тестирования совсем иная. В самом начале работы, когда только создан прототип будущей системы, он тестируется, после чего найденные ошибки исправляются. А затем прототип тестируется опять. При этом опытность дизайнера проявляется исключительно в уменьшении количества итераций. Соответственно, тестирование должно идти параллельно со всеми остальными операциями.

Заключение

Знающие люди утверждают, что путь в тысячу лье начинается с одного сяку. Дочитав до этой страницы, вы это сяку прошли. Важная часть пройдена, но впереди ещё очень большой путь. На нем вы будете предоставлены сами себе, так что я считаю разумным дать вам три небольших совета.

Тестируйте

Без тестирования эффективный интерфейс получить практически невозможно. Это вы уже знаете, но вы ещё не знаете самого главного: тестируя, вы многократно быстрее научитесь проектировать интерфейсы. Одно дело узнать о типичной ошибке из, например, конференции, и совсем другое – обнаружить её в своей работе. Вероятность того, что вы её не повторите, найдя её у себя, гораздо больше, нежели если вы о ней просто узнаете.

Не забывайте о здравом смысле

В дизайне интерфейсов не так уж много аксиом. По сути дела, помимо ограничений человеческого материала, таких, например, как закон Фитса (см. «Быстрый или точный» на стр. 10), которые совершенно объективны и неизменяемы, ничего однозначного нет. Практически любая задача может быть решена многими разными способами, при этом каждая эвристика (включая *все* эвристики из этой книги) при определенных ситуациях могут оказаться ложными. В таких условиях нет ничего более полезного, чем здравый смысл. И тестирование.

Читайте книги

Существует значительное количество литературы, посвященной проектированию пользовательских интерфейсов. Большая часть, как и в любом литературном жанре, состоит их книг плохоньких, но меньшая часть очень хороша. К сожалению, очень малый процент хорошей части (и немалый процент плохой, как и всегда) доступен сейчас на русском языке. Проблема усугубляется тем, что специализированные иноязычные книги трудно добыть в России. К счастью, есть Amazon.

Я особенно рекомендую следующие книги:

- **Donald Norman. The Design of Everyday Things (Currency/Doubleday, 1990)**
  На примерах дверных ручек и прочих мелочей убедительно и понятно излагаются психологические аспекты дизайна. По мнению большинства дизайнеров ПИ, это главная книга, которую нужно прочесть.
- **Jakob Nielsen. Usability Engineering (Morgan Kaufmann Publishers, 1994)**
  В свое время эта книга сделала Якоба Нильсена знаменитым. И не зря.
- **Alan Cooper. About Face: The Essentials of User Interface Design (Hungry Minds, 1995)**
  Книга, соединяющая исключительную свободу мышления с глубиной изложения. Чтение её без труда перестраивает сознание: из обычного обывательского получается сугубо дизайнерское. Очень рекомендуется программистам, поскольку автор не жалеет эпитетов, ругая современное ПО. Получается это у него крайне убедительно.
- **Jeffrey Rubin. Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests (John Wiley & Sons, 1994)**
  Несмотря на то, что основные виды тестирования, вообще говоря, настолько просты, что для их проведения достаточно в меру смыш-

леного ребенка, есть полезные виды тестирования, которые проводить достаточно трудно. К тому же, помимо тестирования, есть ещё и анализ результатов, с которым отнюдь не всё просто.

- **Роберт Солсо. Когнитивная психология (Тривола, 1996)**
  Вводная книга в когнитивную психологию. С одной стороны, наука, с другой стороны, написано доходчиво и просто.

Операция «Человечность и милосердие»

Закончить эту книгу я хочу следующим соображением. Дизайн пользовательского интерфейса, как и вообще любой дизайн, предназначен для того, чтобы делать мир лучше. Вдобавок, в отличие от многих других видов дизайна, дизайн интерфейса предназначен не только для улучшения абстрактного мира: он должен улучшать жизнь обитателей этого мира, т. е. людей. Дизайнер интерфейсов несет двойную ответственность перед миром и перед самим собой, взамен (пока) он получает определенные шансы на славу.

Современные пользовательские интерфейсы слишком часто оказываются ужасными. Исключения из этого правила тоже не идеальны: нет сомнения, что интерфейсы, которые сейчас мы считаем хорошими, с развитием науки окажутся примитивными и неэффективными. Мы всё ещё находимся в начале пути к истинно совершенным интерфейсам. Те из нас, кто пойдет по нему с желанием сделать жизнь пользователей легче и приятнее, с готовностью изменить текущее положение вещей, с отсутствием пиетета к существующим стандартам – изобретут новые интерфейсы и попадут в энциклопедии.

Я сам хочу попасть в энциклопедию. Надеюсь увидеть в ней и ваши имена. До встречи.

Предметный указатель

П

Р

Э

A-Z