Руководство по UI дизайну для программистов

Джоэл Сполски

Автор: Джоэл Сполски
Переводчик: Наталья Лунева
Техническая поддержка и моральная помощь: Алексей Матюшкин
Редактор: Евгений Дурцев
10. 4. 2000

Глава1: Все под контролем, или баллада о счастливых пользователях

Большинство известных мне программистов, работающих на C++, с большой опаской относятся к созданию пользовательских интерфейсов (UI). Меня это, признаться, удивляет, поскольку программирование UI, на мой взгляд,-- дело простое, очевидное и увлекательное.

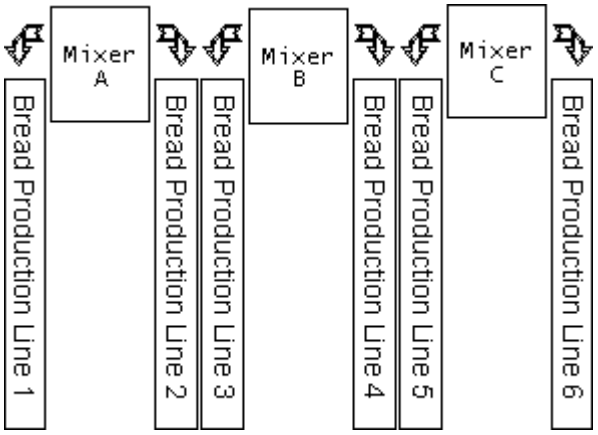
Простое -- потому, что самый сложный алгоритм, который вам может потребоваться,-- алгоритм отцентровки одного прямоугольника в другом. Очевидное -- потому, что, сделав ошибку, вы можете ее немедленно увидеть и исправить. Увлекательное -- потому, что вы можете сразу же увидеть результаты вашей работы. Работа по UI дизайну сродни работе скульптора: вы непосредственно ваяете программу.

Мне кажется, страх программистов перед UI программированием объясняется их страхом перед UI дизайном. Они полагают, что UI дизайн похож на дизайн графический: мистический процесс создания классного, необъяснимо художественного материала, затеянный креативными, одетыми во все черное, украшенными замысловатыми пирсинг-узорами людьми, небрежно потягивающими абсент через трубочки. Себя же программисты оценивают как логически мыслящих аналитиков: блестящее умение аргументировать, слабовыраженный художественный вкус. К тому же, пьют они растворимый черный кофе. И поэтому работать над UI дизайном они не могут.

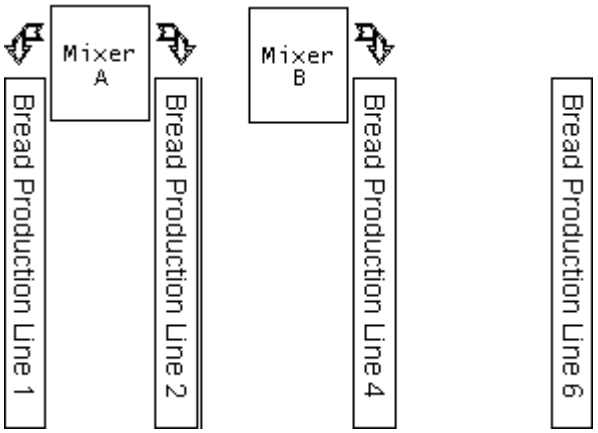
На самом деле, UI дизайн достаточно прост и рационален. Это не мистическое действо, для успешного выполнения которого необходим диплом об окончании художественной школы и хаос ярко оранжевых волос на макушке. Существует рациональный подход к пользовательским интерфейсам с набором простых, логичных правил, которые можно применять всякий раз, когда возникает необходимость улучшить интерфейс программ, над которыми вы работаете.

Я не собираюсь читать вам проповедь на тему "Дзен и искусство UI дизайна". Это вовсе не искусство и не буддизм. Это просто набор правил. Рациональный и методический подход. И создана эта книга для программистов. Это значит, что я исхожу из того, что вам не нужны инструкции на тему, как создать панель меню. Скорее, вы размышляете о том, что вам выложить на эту панель (или нужна ли она вам вообще). Я познакомлю вас с аксиомой дизайна, которая лежит в основе любого хорошего UI дизайна, и понять ее смысл -- дело не такое уж и сложное.

Свою рабочую карьеру я начал на большой промышленной хлебопекарне. На хлебопекарне было установлено шесть линий по производству хлеба. Каждые две линии были оснащены одним тестосмесителем, рассчитанным на производство 180 килограммовых кусков теста, которое могло направляться на левую или правую линии:



По крайней мере, так оно выглядело на бумаге. На самом деле, смеситель **C** еще не был установлен, так же как третья и пятая линии. Поэтому, все производство выглядело так:



Внимательные читатели уже могли воскликнуть: "каким же образом тесто из смесителя **B** попадало на шестую линию?" Так вот, как раз в этот момент на сцене появляется крошка Джоел. Можете этому верить или нет, но моя работа заключалась именно в том, чтобы стоять по левую сторону от смесителя **B**, ловить с помощью огромного чана на колесах гигантские 180 килограммовые куски вылетающего из смесителя теста, перетаскивать чан к 6-ой линии и вываливать на нее тесто. И так каждые 10 минут, с 10 часов вечера до 4 утра.

В работе существовали и дополнительные трудности. Шестая линия, в действительности, не могла сразу справиться со 180 килограммами теста, поэтому мне приходилось резать его огромным ножом на десять кусков. Я даже не хочу описывать, насколько абсурдно тяжелой была эта работа.

Понятно, что в течении первых нескольких дней у меня не получалось ничего. Работа казалась невыполнимой. Каждая клеточка моего тела ныла и обливалась слезами. У меня болели даже те места, о существовании которых я не подозревал.

Сначала я просто не мог постоянно наполнять шестую линию тестом. Каждый раз, когда поток теста прерывался, на конвейре образовывалась пустота. Пустота вкатывалась в печь, печь (поддерживающая постоянный уровень энергии даже при уменьшенном количестве теста) перегревалась, хлеб подгорал. Иногда шестая линия останавливалась, но смеситель в том же темпе продолжал выбрасывать тесто, и я уже не знал, где найти очередной чан для хранения теста. Когда такое происходило, мне приходилось мыть и смазывать маслом пол для того, чтобы сначала швырять тесто на пол, а затем его отдирать. Не могу сказать, что это был оптимальный вариант, поскольку пролежавшее больше получаса тесто перекисало, и испеченный из него хлеб был, м-м-м... ну, не самым вкусным. Отодранное от пола тесто приходилось рубить на пятикилограммовые куски и класть их по отдельности в смесь для следующей порции.

Спустя неделю я уже поднаторел настолько, что, если мне не изменяет память, из каждого 10-минутного цикла урывал две минуты для отдыха. Я разработал точную процедуру и понял, что делать со смесителем, когда линия останавливалась.

И у меня появилось время, чтобы поразмышлять о том, почему некоторые дни оказываются более удачными чем другие.

И в один прекрасный день, предаваясь подобным размышлениям, я заметил, что у одного из чанов плохо поворачиваются колеса. Иногда этот чан отказывался катиться в нужную мне сторону, и я врезался в какую-нибудь фиговину, стоящую на пути. Это вызывало легкое неудовольствие. В другой раз, когда я подтягивал цепь, чтобы приподнять чан, я царапался -- совсем чуть-чуть - о мелкую металлическую зазубрину на цепи. Еще одно маленькое расстройство. В другой раз, когда я бежал с пустым чаном к смесителю, чтобы подхватить очередное извержение теста, я подскальзывался на капле разлитого на полу масла. Нет, я не падал, заметьте, но -- очередной маленький, крохотный повод для расстройства.

Бывали и другие дни -- дни маленьких побед. Я научился самым точным образом определять время производства теста, так, что свежее тесто появлялось на несколько секунд раньше того, как заканчивалась порция теста на линии печи. Это было гарантией выпечки лучшего хлеба из свежайшего теста. Бывали победы и того меньше: я замечал, как кусочек теста вылетал из смесителя и прилеплялся к стене, и я тут же -- хоп! -- небольшим мастерком, который всегда был у меня в заднем кармане брюк, снимал его со стены и бросал в мусорный бак. Йес! Или, разрезая тесто на куски, я просто чувствовал, как *легко* и *свободно* идет нож. Краткие моменты удовлетворения, когда мне удавалось контролировать окружающий меня мир, пусть даже и самым незначительным образом.

Вот так и проходили мои дни. Пучок маленьких расстройств, букетик маленьких побед. Но все они суммируются. Даже, на первый взгляд, незначительное, единичное событие оказывает влияние на наше настроение. Наше эмоциональное *я* не интересуют количественные характеристики явления, только его качество.

И так я стал понимать, почему некоторые дни приносили большее удовлетворение, чем другие: эти дни были наполнены большим количеством маленьких побед и малым количеством мелких неудач.

Спустя несколько лет, уже учась в колледже, я познакомился с психологоческой теорией доктора Мартина Е. П. Селигмана (Dr. Martin E.P. Seligman) под названием "Приобретенная беспомощность" (Learned Helplessness). Центральный тезис этой теории, подтвержденный годами исследований, гласит: состояние депрессии часто вырастает из чувства *беспомощности*, когда человек ощущает, что не может контролировать происходящее.

Уверенность в том, что вы контролируете происходящее, в том, что ваша работа результативна, напрямую связано с чувством удовлетворения. Злость, разочарование, огорчение сопровождают вас, когда что-то, пусть даже и незначительное, выходит из-под вашего контроля. На вашей клавиатуре западает клавиша "пробел". Вы печатаете и замечаете, что некоторые слова написаны слитно. Вы нажимаете "пробел", еще раз, еще раз -- и *ничего не происходит*. Раздражение и злость нарастают. Что-то случилось с ключом к двери подъезда: он заедает каждый раз, когда вы пытаетесь его повернуть. И вы вновь раздражаетесь. Подобные мелочи накапливаются,-- неудовлетворенность повседневностью растет. Они кажутся слишком пустячными, чтобы о них задумываться (ну посудите сами: в Африке люди умирают от голода, куда уж тут огорчаться по поводу каких-то клавиш!), и тем не менее, они влияют на наше настроение.

Оставим на минуту наши психологические измышления и вернемся к компьютерам.

И придумаем типичного пользователя Windows по имени Петя. Помните о том, для кого вы создаете пользовательский интерфейс, -- и ваша работа будет значительно облегчена. Чем большим количеством реальных человеческих качеств вы наделите своего воображаемого пользователя, тем лучше будете осознавать, как он может использовать ваш продукт. Итак, Петя. Он работает бухгалтером в издательстве технической литературы, Windows исползует уже в течение шести лет на работе и немного дома. Компетентен, разбирается в технике. Инсталлирует программное обеспечение на своих компьютерах, почитывает журнал "PC Magazine" и как-то раз программировал несложные Word макросы, чтобы помочь симпатичной секретарше в бюро с рассылкой счетов. Дома у него кабельный модем. Петя никогда не работал с Macintosh. "Они слишком дорогие", -- скажет он вам, -- "компьютер на 700 Мгц с памятью на 128 Мб можно купить за..." Да, Петр, мы поняли. Однажды подруга Петра, Марина, просит его помочь ей с ее новым Macintosh iBook: ей ужасно понравилась эта светящаяся коробочка... Петя вздыхает, садится за компьютер и мрачнеет.

"Ненавижу эти штучки." Наконец-то, он разобрался, что к чему, но его вывод однозначен: "У Маков -- совершенно идиотский пользовательский интерфейс."

Идиотский? Что он несет? Спроси у любого ребенка, и он тебе скажет, как удобен и элегантен интерфейс Macintosh!

В чем же дело? Вот мои предположения.

В Macintosh, если вы хотите передвинуть окно, нужно ухватиться мышью за любой край окошка и потянуть его. В Windows нужно тянуть за панель заголовка. Если же вы потянете за край окна, изменится форма окна. Бедный Петя пытался расширить окно, потянув за правый его край. К его разочарованию, глупое окно просто переместилось.

В Windows, чтобы закрыть диалоговое окно, нужно щелкнуть либо клавишей "enter", либо "пробелом". В Macintosh "пробел" не работает. Зато можно просто кликнуть мышью. В первый раз, когда Петр пытался закрыть диалог, он нажал на "пробел", -- действие, которое он совершенно неосознанно совершал последние 6 лет. Ничего не случилось. Опять-таки не осознавая происходящего, Петя еще раз, уже сильнее, нажал на "пробел": возможно, Macintosh не зарегистрировал его первого нажатия. Так вот, нет -- зарегистрировал, но -- не отреагировал: ему было до лампочки! Пришлось Пете воспользоваться мышью. И нахмурить брови.

Петр знал команду Alt+F4 для закрытия окон. В Macintosh эта команда меняет громкость. Петя хотел кликнуть по иконке Internet Explorer на экране, которая была частично закрыта другим окном. Он нажимает Alt+F4, чтобы закрыть окно, и тут же делает двойной щелчок мышью по иконке. Alt+F4 увеличивает громкость и не закрывает окно, и поэтому двойной щелчок мышью активирует клавишу помощи на панели инструментов окна, которое Петр хотел закрыть, и, соответственно, открывает окно помощи. Петя, который просто хотел закрыть мешавшее ему окно, сидит теперь перед двумя открытыми и совершенно ненужными ему окнами.

И брови его сдвигаются еще сильнее. К концу дня они вытянуты в одну гневную линию. Компьютер ему не подчиняется. Ни клавиша "пробел", ни Alt+F4 не работают, как если бы они вышли из строя. Окна его не слушаются: перемещаются, когда он хочет их расширить. На подсознательном уровне чувство потери контроля переходит в ощущение беспомощности, и затем в состояние неудовлетворенности. "Мне нравится мой компьютер",-- говорит Петр, -- "на нем все настроено так, чтобы мне было удобно на нем работать. Эти Маки неуклюжи и неудобны в использовании. Им следовало бы сделать нормальную операционную систему под этим красивым логотипом, а не разбрасываться яблоками в подворачивающихся Ньютонов все эти годы. Полный облом!"

Петр прав, уж мы-то знаем. Его ощущения возникли вопреки тому факту, что Macintosh, на самом деле, прост в использовании -- для пользователей Macintosh. Нет общего правила, которое определяет, по какой команде закрывается окно. Программисты Microsoft, которые предположительно копировали интерфейс Macintosh, возможно считали, что они добавляют классную функцию, которая позволяет изменить размер окна, когда вы тянете за его край. Программисты MacOS 8.0 скорее всего были убеждены, что они добавляют классную функцию, которая позволяет вам перемещать окно, когда вы тянете за его край.

Большинство дебатов по поводу пользовательских интерфейсов совершенно излишни. Windows лучше, потому что они предлагают больше способов менять размер окна. Ну и что? Суть-то не в этом. Суть в следующем: реагирует ли пользовательский интерфейс так, как пользователь того *ожидает*? Если нет, пользователь будет ощущать собственную беспомощность и невозможность контролировать ситуацию, то же, что чувствовал я, когда колеса моего чана для теста не поворачивались в нужную мне сторону, и я врезался в стену. Бум.

Пользовательский интерфейс очень важен, поскольку он влияет на чувства, эмоции, настроение пользователя. Если дизайн неадекватен, пользователь чувствует, что он не может контролировать созданное вами программное обеспечение, он будет недоволен и *несчастен*, в буквальном смысле, и виновато в этом будет ваше программное обеспечение. Если же дизайн удобен и работает так, как пользователь того ожидает, он будет в хорошем настроении претворять свои маленькие цели в жизнь. Ого! Я залил CD! *Получилось!* Классная программа!

Чтобы люди чувствовали себя счастливыми, нужно дать им возможность ощущать, что ситуация находится под их контролем. Для этого вам нужно уметь *правильно* интерпретировать их действия. Пользовательский интерфейс должен вести себя так, как этого ожидают пользователи.

Итак, основная аксиома UI дизайна гласит:

> **Хороший дизайн пользовательского интерфейса подразумевает, что программа соответствует ожиданиям пользователей о том, как она должна себя вести.**

Все остальное -- следствия.

Руководство по UI дизайну для программистов
Глава2: Как узнать, чего они ждут

Когда новый пользователь приступает к работе с новой программой, его голова наполнена опытом прошлых встреч с компьютером. У него есть определенные ожидания по поводу того, как новая программа будет работать. Если он уже использовал подобное программное обеспечение, он будет думать, что эта новая программа будет работать похожим образом. Если он просто использовал какое-либо программное обеспечение, он будеть думать, что новая программа соответствует каким-то общим определенным условностям. У него даже могут быть совершенно разумные мысли о том, как будет работать интерфейс данной программы. Все это называется *моделью пользователя*: представление о том, для чего и как программа будет работать.

Программа тоже обладает "ментальной моделью", которая, в отличие от модели пользователя, закодирована в биты и самым последовательным образом выполняется CPU. Название ей -- *модель программы*, и она есть -- **Закон**. Как мы узнали в Главе 1, если модель программы соответствует модели пользователя, у нас получился удачный пользовательский интерфейс.

Рассмотрим пример. Когда вы добавляете картинку в документ Microsoft Word (или любого другого текстового редактора), картинка сохраняется в том же файле, что и сам документ. Вы можете создать картинку, вставить ее в документ, *удалить оригинальный файл с картинкой*,-- voilá -- картинка сохранится в документе.

Возьмем HTML, который не позволяет вам этого. Картинки в HTML должны храниться в отдельных файлах. Если посадить перед симпатичным WYSIWYG HTML-редактором (например, *FrontPage*) человека, который до этого пользовался только текстовыми редакторами и ничего не знает о HTML, он совершенно точно будет ожидать, что его картинка будет сохранена в самом документе, а не в отдельном файле. Можете назвать это *инерцией модели пользователя*.

В итоге мы получаем конфликт между моделью пользователя (картинка будет сохранена в документе) и моделью программы (картинка должна быть сохранена в отдельном файле). Дизайн интерфейса станет источником проблем.

Если вы пишите программу наподобие FrontPage, вы только что нашли первую проблему своего дизайна. Изменить HTML вы не можете. Но вам придется придумать что-то, что приведет модель программы в соответствие с моделью пользотеля.

У вас есть выбор. Вы можете попытаться изменить модель пользователя. Что окажется невероятно сложным. Вы могли бы объяснить все в руководстве, но всем известно, что пользователи руководств не читают, да, наверное, и не обязаны. Вы могли бы создать всплывающее диалоговое окно с сообщением, что картинка в документе сохранена не будет. Что вызовет *две* дополнительные проблемы: во-первых, диалоговые окна раздражают продвинутых пользователей, во-вторых, диалоговые окна никто не читает (подробнее об этом в Главе 6).

Что ж, если гора не идет к Магомету... Практически всегда лучшим решением будет изменить модель программы, а не модель пользователя. Например, по добавлению картинки, ее копия помещается в под-директорию файла, что соответствует представлению пользователя о том, что картинка копируется (а оригинальный файл может быть удален).

Как познать модель пользователя?

В целом, это элементарно. Спросите их! Выберите наугад 5 человек с работы, или друзей, или родственников, расскажите им в общих словах о назначении вашей программы ("программы для создания веб-страниц"). Затем опишите ситуацию: "Ты работаешь над веб-страничкой. У тебя есть файл с картинкой под именем Picture.JPG. Ты добавляешь картинку на свою страницу." Далее, задайте пару вопросов и попробуйте составить мнение об их модели пользователя. "Куда делась картинка? Если ты удалишь файл Picture.JPG, сможет ли страница показывать твою картинку?"

Один мой друг работает над приложением "Фотоальбом". После того как вы добавили фотографии, приложение показывает вам набор *ярлычков* (thumbnails): крохотные копии всех фотографий. Их генерация занимает массу времени, особенно если вы добавляете много фотографий, поэтому мой друг хочет складывать все ярлычки куда-нубудь на жесткий диск так, чтобы генерация всех проходила одновременно. Сделать это можно по-разному. Например, сложить все в один файл под именем Thumbnails. Или сохранять их в отдельных файлах, но в одной под-директории Thumbnails. Или маркировать их как скрытые файлы в операционной системе так, чтобы пользователь о них не знал. Мой друг выбрал иной путь, который, по его мнению, был оптимальным: он поместил ярлычок каждой картинки picture.JPG в новый файл, названный picture_t.JPG, в той же директории. При создании альбома с 30 фотографиями вы получаете 60 файлов в директории, включая ярлычки картинок.

Можно неделями дискутировать о преимуществах и недостатках того или иного способа хранения картинок, но есть гораздо более научный способ найти оптимальный вариант. Спросите пользователей, где, по их мнению, хранятся ярлычки картинок. Безусловно, некоторые из них не имеют ни малейшего представления и никогда об этом не задумывались, другим все равно, но когда вы соберете много различных мнений, вы сможете увидеть решение, которое вас устроит. Самый часто встречающийся вариант ответа и есть лучшая модель пользователя, и только от вас зависит, насколько удачно вы совместите обе модели.

Следующий этап -- проверка ваших предположений. Постройте модель или прототип вашего интерфейса и дайте нескольким людям задания. Попросите их комментировать свои действия в то время, как они решают поставленную задачу. Ваша цель заключается в том, чтобы понять, чего они ожидают. Предположим, вы дали задание: "вставить картинку". Если вы увидите, что человек пытается мышью затащить картинку в документ, вы поймете, что вам следует поддержать технологию *drag-and-drop*. Если он остановит курсор на кнопке "*Вставка*" на панели инструментов, вы осознаете, что было бы полезно разместить в этом меню опцию "*Картинка*". Когда же он на панели инструментов попытается заменить слова "*Times New Roman*" на "*Вставить картинку*" , вы достаточно быстро сообразите, что вам попался реликтовый экземпляр, который, будучи незнаком с графическими интерфейсами, ищет командную строку.

Какое количество пользователей следует привлекать к подобным тестам? Инстинкт может подсказать вам ответ "чем больше, тем лучше", что имеет смысл, если вы проводите научный эксперимент. В любом другом случае инстинкту доверять не стоит, но стоит прислушаться к людям, которые занимаются usability тестированием профессионально. Они советуют ограничить число пользователей до пяти - шести. Если вы привлечете большее количество народу, то увидите повторяемость результатов, и ничего большего, чем несколько часов потерянного времени, не приобретете.

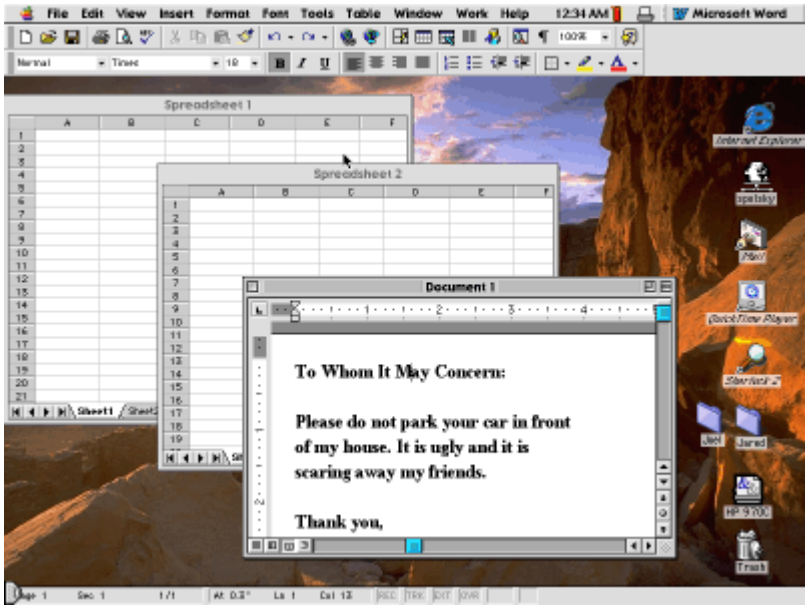
Для такого тестирования не требуется официальная лаборатория или привлечение "людей с улицы". Вы просто, с доброй улыбкой на лице, подходите в первому встречному и спрашиваете его, а не могли бы вы помочь мне провести usability тестирование. В разговоре с ним не растекайтесь мыслью по древу и не командуйте. Просто попросите его думать вслух, задавайте вопросы, которые предполагают альтернативные варианты ответов, и попытайтесь понять его модель пользователя.

Если ваша модель программы нетривиальна, она, скорее всего, не будет соответствовать модели пользователя.

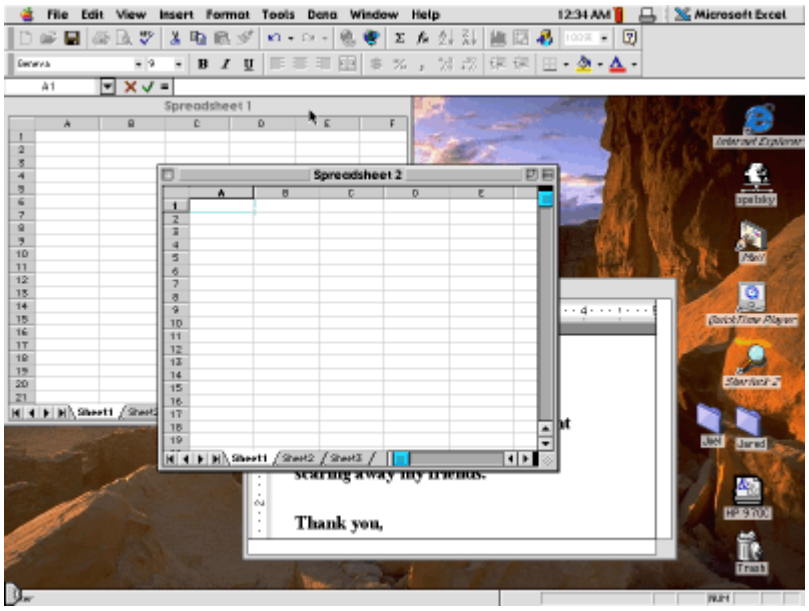
Давным-давно, когда мне было лет шесть, папа принес домой один из первых карманных калькуляторов. Он пытался меня убедить, что внутри его находится *компьютер*. Я подвергал его слова сомнению. Все компьютеры, которые я видел в *Star Treck*, были размером с комнату и имели огромные катушечные магнитофоны. Поэтому я считал, что некое умное соотношение между кнопками на калькуляторе и знаками на его дисплее обеспечивало математически правильные результаты. (Ну ладно, хватит хихикать, мне было всего шесть лет).

Важно помнить, что модели пользователей, как правило, не очень сложны. Догадки обычных пользователей о том, как работает та или иная программа, будут скорее простыми, очевидными и будут отличаться от хитроумных задумок программистов.

Откройте два документа Excel и один Word документ на своем Macintosh. Практически любой новичок догадается, что открытые окна друг от друга не зависят. Три независимых окна, не так ли?



Модель пользователя предполагает, что щелчок мышью по Таблице 1 переместит это окно на передний план. На самом же деле, -- сюрприз! -- на переднем плане окажется Таблица 2. Сюрприз, надо признать, для большинства неприятный.



Оказывается, модель программы *Microsoft Excel* подразумевает буквально следующее: "Для каждого приложения существуют *специальные невидимые полотна*, к которым приклеиваются все открытые окна данного приложения. Таким образом, когда вы выводите одно окно *Excel* на передний план, там же оказываются и все остальные."

Аха... Невидимые полотна... А теперь прикиньте, чему равняется вероятность того, что модель пользователя включает в себя концепцию невидимых полотен? Правильно, стремится к нулю. Вполне естественно, что такое поведение окон нового пользователя как минимум удивит.

Еще один пример из мира *Microsoft Windows*: комбинация клавиш *Alt+Tab* переключает вас на другое окно. Большинство пользователей, скорее всего, предположат, что одновременное нажатие этих двух клавиш обеспечивает ротацию между всеми открытыми окнами. Например, при открытых окнах **А**, **Б**,**В** и активированном **А**, *Alt+Tab*перенесет вас к окну **Б**. Следующее нажатие *Alt+Tab* активирует окно **В**. Как бы не так! Второе нажатие *Alt+Tab* возвращает вас к окну **А**. Единственный способ добраться до **В** -- держать клавишу *Alt* нажатой и два раза щелкнуть *Tab*'ом. Таким образом, *Alt+Tab* предоставляет хорошую возможность путешествовать между **А** и **Б**, но догадаться, как добраться до **В** -- почти невозможно, потому что данная модель является более сложной, чем модель ротации между всеми открытыми окнами.

Приспособить модель программы к модели пользователя -- нелегкое дело, даже когда модели простые. Оно становится практически невыполнимым, когда сложность моделей возрастает. Поэтому идеальный вариант -- использовать простейшую модель из всех, которые кажутся вам возможными.

Руководство по UI дизайну для программистов
Глава3: Бремя выбора

Когда вы заходите в ресторан и видите знак "Вход с собаками воспрещен", вы возможно, сочтете этот знак чисто запрещающим: господин владелец ресторана не любит собак, и поэтому, открыв свое заведение, повесил такой знак.

Если бы дело было *исключительно* в этом, повсюду бы красовались знаки "Вход со змеями воспрещен" -- в конце концов, змей не любит никто. Или "Вход слонам воспрещен" -- они же разломают все стулья, когда попробуют на них усесться.

На самом деле, причина появления таких знаков -- историческая: они указывают на то, что люди раньше частенько пытались приходить в рестораны с собаками.

Большая часть запрещающих знаков появляется тогда, когда владельцы заведений, намаявшись с людьми, которые делают нечто, их не устраивающее, вывешивают знак "нечто -- не делать!". Если вы заглянете в старинные семейные забегаловки, открытые лет пятьдесят назад, то увидите там стены, исписанные воззваниями типа: "Просьба не класть ранцы на прилавок!" -- и это скорее антропологическое свидетельство того, что посетители раньше просто-таки злоупотребляли возможностью положить ранец на прилавок. Возраст подобного знака поможет вам также уточнить, в каких годах ранцы были популярны среди местного студенчества.

Иногда историческую подоплеку выявить сложнее. Знак "Пожалуйста, не приносите в парк стеклянные бутылки" очевидно подразумевает, что кто-то когда-то в этом парке порезался о разбитое стекло, гуляя босиком по лужайке, и -- хотите пари? -- выставил счет городскому муниципалитету.

Подобные археологические свидетельства можно найти и в программном обеспечении. Название им "*Диалог редактирования параметров*". В разделе меню *Tools* найдите категорию *Options*, откройте ее и вашему взору представится задокументированный спор дизайнеров программы: надо ли автоматически открывать последний файл, над которым работал пользователь? -- да! -- нет! Спор, который растягивается на недели, каждая сторона щадит чувства другой строны, и программист вписывает в код строку #ifdef как акт самообороны в то время, как дизайнеры доказывают свою правоту. В конце концов, обе стороны соглашаются оставить право выбора за пользователем и ввести предмет спора в раздел опций.

Подобные споры разгораются не только внутри компаний, но и внутри каждого из нас. "Оптимизировать базу данных под скорость или под размер? Под размер? Под скорость? ..." Ваш ли внутренний это спор, или дебаты в вашей команде, результаты в той или иной степени похожи на то, что с полным правом можно назвать самым маразматичным диалогом в истории операционной системы *Windows*. Диалог этот по своей тупости заслуживает награды. Целой категории наград. Имя его -- *Find Setup Wizard*.

Появляется он, когда вы хотите найти что-либо в файле справки:

Проблема первая: он отвлекает. Вы пытаетесь найти *помощь* в файле *помощи*. В этот конкретный момент вам абсолютно все равно, маленькая ли база данных, большая, оптимизирована ли она под нужды клиента или покрыта шоколадом. Вам нужна *помощь*, а этот отвратительный диалог сообщает вам, что он должен создать список или базу данных. И педантично читает лекцию на целых три параграфа, смысл которых приводит вас в недоумение. Там еще есть ужасно нелепая фраза "ваш(и) файл(ы) помощи". Видите ли, у вас может быть *один или более* файлов. Как будто бы вам в этот момент было важно, один у вас файл или несколько. Как будто бы это могло на что-то повлиять... Но вот программист, который работал над диалогом помощи, был, вероятно, потрясен до глубины души возможностью создать один или более файлов, и утаить это от вас, написав просто "файл помощи", было бы верхом вероломства, не так ли?

Я даже не хочу говорить о том, что большинство людей, обращающихся к помощи, просто не понимают ваших иероглифов. Или о том, что даже продвинутые пользователи, программисты с кандидатской степенью, которые знают все о построении оптимизационных бинарных древовидных структур при индексировании текстов,-- не смогут понять, а какой, собственно, выбор им предлагают совершить.

И, напоследок, чтобы присыпать обиду солью оскорбления: это -- даже не окно. Это -- *wizard* (парафраз второй страницы которого может звучать приблизительно так: "Благодарим Вас за то, что Вы по доброй воле потратили Ваше время на полную ерунду!"). При всем при этом: очевидно, что у программиста были *свои* представления на счет, какая из опций является предпочтительней; иначе бы он не стал утруждать себя рекомендацией одной из них.

Отсюда вытекает второе правило UI дизайна:

> **Каждый раз, предлагая опцию, вы просите пользователя сделать выбор.**

Сама по себе идея, просить пользователя сделать выбор, не так уж и плоха. Право выбора гарантировано конституцией и иногда даже доставляет удовольствие. Например, люди обожают заказывать напитки, приготовленные на основе эспрессо, потому что им предоставляется тако-о-ой выбор. Будьте любезны, большую порцию, без кофеина, с однопроцентым молоком, с привкусом карамели и взбитыми сливками. В высоком стакане!

Проблемы возникают, когда вы просите их сделать выбор, который их *ничуть не волнует*. Как в примере с файлами помощи: человек открывает справку, надеясь решить проблему, которая у него возникла, когда он пытался решить какую-то *важную для него* задачу, скажем, нарисовать открытку -- приглашение на день рождения. К сожалению, решение этой задачи в какой-то момент застопорилось из-за того, что он не смог сообразить, как можно распечатать воздушные шарики вверх ногами. Итак, он вынужден отвлечься и открыть файл помощи. И тут деятельный программист Microsoft -- разработчик индекса помощи с навязчивой идеей значимости собственной фигуры в истории программирования -- берет на себя смелость прервать нашего пользователя *еще раз* и поучить его созданию связанных списков или баз данных. Вот этот второй уровень вмешательства не имеет никакого отношения к созданию-приглашения-на-день-рождения и гарантированно приведет пользователя в смущение, раздражение и может даже заставить его плюнуть на незаконченную работу и купить открытки в соседнем киоске.

Уважаемые программисты, поверьте, пожалуйста: пользователей заботит гораздо меньшее количество вещей, чем вы думаете. Ваши программы они используют для решения *своих* задач. И все, что их волнует, -- это решение их задач. Если речь идет о графической программе, они захотят контролировать каждый отдельный пиксел, чтобы добиться наивысшей чистоты деталей. Если в их руках программа для изготовления веб-страниц, будьте уверены, они лягут костьми для того, чтобы их страница выглядела точно так, как они ее себе представляют.

Но: они не задумаются о том, находится ли панель инструментов вверху или внизу экрана, индексирован ли раздел помощи или нет. Они безразличны ко многим вещам. И ответственность дизайнеров как раз в том и заключается, чтобы освободить пользователей от необходимости принимать подобные решения. Сваливание ответственности за принятие подобных решений на пользователя -- есть вершина высокомерия программиста, который не дал себе труда додумать до конца, какое же решение в данном случае будет оптимальным. (Тем более, когда вы, пытаясь прикрыть собственное неумение принимать решения, подсовываете пользователю какого-то нелепого советчика -- типа диалога помощи, как это сделала команда WinHelp.

Как будто бы пользователь -- идиот, которого вы посредством пошагового мини-курса о предлагаемых опциях хотите научить принимать *научно обоснованные* решения.)

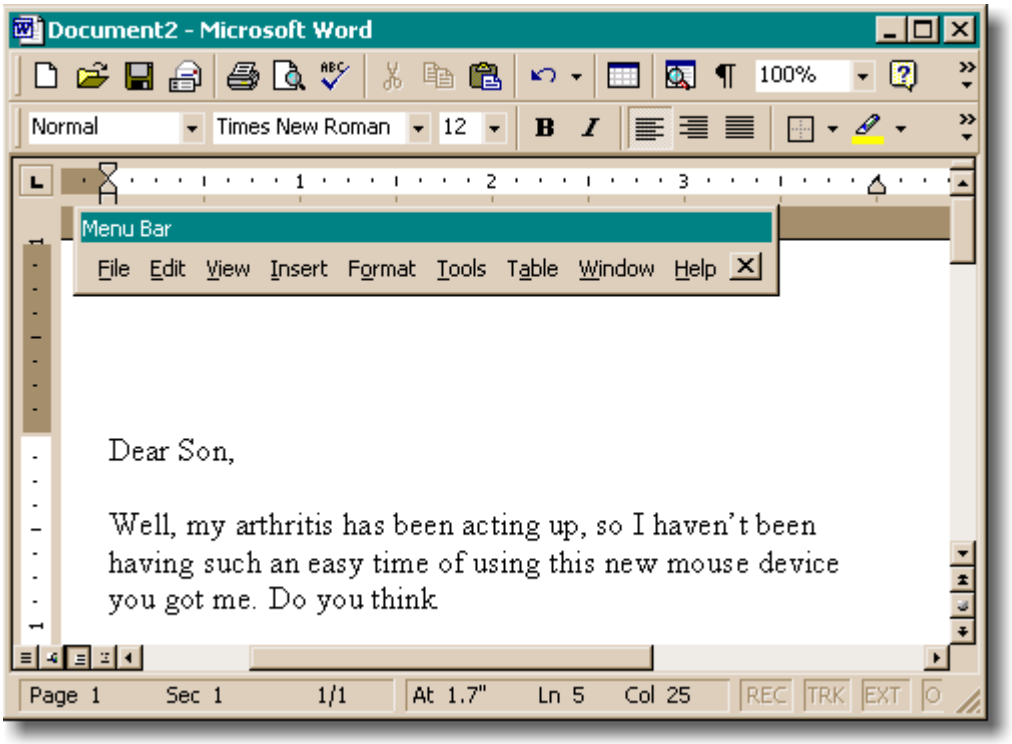
Вы уже, должно быть, слышали о том, что дизайн -- это искусство *принятия решений*. Когда вы разрабатываете дизайн контейнера для мусора, вам приходится принимать решения о том, как связать воедино конфликтующие требования. Контейнер должен быть тяжелым, чтобы он не улетел при порыве ветра. И легким, чтобы мусорщик мог легко его опорожнить. Он должен быть большим, чтобы вмещать большое количество мусора. И маленьким, чтобы не стоять у людей на дороге. И если вы -- дизайнер -- пытаетесь передать ответственность пользователю, ставя его перед выбором, вы плохо делаете свою работу. Ваш конкурент создаст более легкую в использовании программу, которая помогает решать те же самые задачи с меньшим количеством ненужных отвлечений, и покупатель уйдет от вас и будет радоваться.

Появившийся в 1990 году *Microsoft Excel 3.0* был первым приложением, в котором использовалась панель инструментов. Вещь оказалась нужной, всем понравилась, все ее скопировали; приложение без панели инструментов сейчас -- явление достаточно редкое.

Панель инструментов оказалась настолько удачной находкой, что разработчики *Excel* решили провести исследование, какие из команд на панели использовались наиболее часто. Создали специальную версию, которая вела статистический подсчет и сообщала данные Microsoft'у, раздали публике. Следующая версия *Excel* вышла с *дополнительной* панелью инструментов, на которой находились пиктограммы наиболее часто используемых команд. Великолепно.

Проблема была в том, что команда разработчиков панели инструментов не смогла остановиться. Они захотели, чтобы вы могли оптимизировать панель инструментов под свои нужды. Чтобы вы смогли перетащить ее туда, куда вам захочется. Потом они решили, что *панель меню* -- это та же славная панель инструментов, только со словами вместо иконок, и теперь ее тоже можно перетаскивать туда, куда захочется. Быстрее, выше, сильнее! Но почему-то это новшество никого не заинтересовало. Никто не захотел ничего никуда перетаскивать, все были довольны панелями меню и инструментов вверху экрана.

Но вот осталась одна закавыка: если вы хотите открыть подменю "Файл" в меню и случайно зацепите панель чуть левее чем нужно, вся она перекочует как раз туда, куда бы вам меньше всего хотелось: в тело вашего документа.

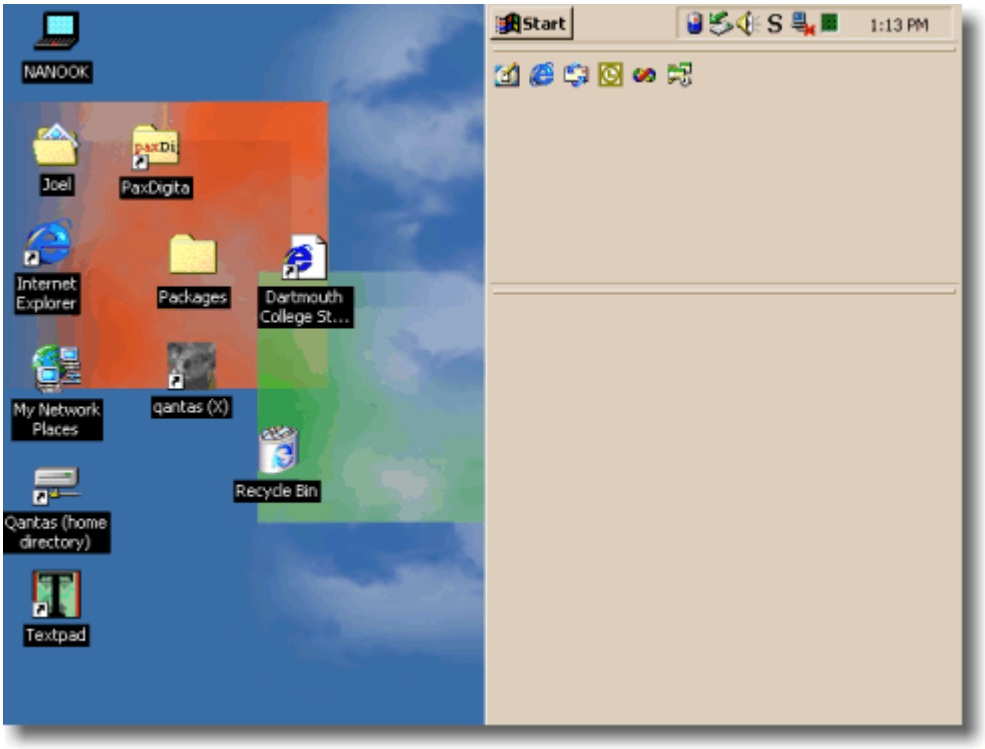


Сколько раз вы уже с этим сталкивались? И самое неприятное: если вы сделали это случайно,-- вы не знаете, каким образом вы этого добились или как вернуть панель на место. В итоге у нас есть никому не нужная (кроме, пожалуй, сотой доли процента пользователей) опция -- перемещение панели меню -- которая мешает работе почти каждого из нас.

Однажда мне позвонила знакомая. У нее что-то не получалось с отправкой почты. "У меня половина экрана серая", -- сказала она.

Половина экрана -- серая?

Минут через пять телефонного разговора я сообразил, что произошло. Она случайно переместила панель инструментов на правую половину экрана, а затем -- случайно -- расширила ее:



Намеренно подобные вещи не делает никто. И многие пользователи компьютеров не в состоянии решить внезапно возникшую проблему; по определению: если вы случайно переконфигурировали одну из опций программы,-- вы, скорее всего, не знаете, как вернуть ее в прежнее состояние. Количество людей, которые вынуждены де-инсталлировать программное обеспечение, чтобы потом проинсталлировать его заново, потому что оно как-то не так себя ведет, слегка шокирует. Но они, по крайней мере, знают, как сделать это. (Они научились де-инсталлировать сначала, потому что иначе все параметры сохранят свои ошибочно измененные значения.)

"Но послушайте, опции необходимо сохранить для *продвинутых*пользователей, которые с удовольствием меняют настройки своих систем!" -- наверняка возразите вы. По правде говоря, это не так уж и необходимо, как вам кажется. Моя собственная история, к примеру. Я пытался сменить свою старую раскладку клавиатуры на более продвинутую -- *Dvorak*. Проблема состояла в том, что я пользуюсь не одним компьютером. Я работаю с разными компьютерами, в том числе с компьютерами других людей. Я регулярно работаю с тремя компьютерами дома и с тремя на работе. Я работаю с компьютерами в тестовой лабаратории в офисе. Вы собираетесь оптимизировать системы всех компьютеров, с которыми работаете? По-моему, овчинка выделки не стоит.

Большинство продвинутых пользователей работают регулярно с несколькими компьютерами; каждые пару лет они делают *upgrade*, каждые три недели переинсталлируют операционные системы. Трудно отрицать тот факт, что, когда они *впервые* узнали о возможности полностью переделать раскладку клавиатуры в *Word*е, они тут же переделали всю систему в соответствиями со своими предпочтениями. Но как только они проинсталлировали *Windows 95*, их личные установки потерялись, установки на работе отличались от установок на домашнем компьютере... в общем, они устали регулярно изменять системы и перестали этим заниматься. Я разговаривал со многими своими знакомыми, которых можно отнести к категории продвинутых пользователей; почти никто не оптимизирует системы, они меняют минимальное количество установок, необходимых для того, чтобы система вела себя прилично.

Каждый раз, предлагая опцию, вы просите пользователя сделать выбор. Это значит, что им нужно будет что-то обдумывать и что-то решать. Это не обязательно плохо, но в общем, вам следует всегда минимизировать количество вопросов, по которым пользователь должен принять решение.

Это не значит, что нужно *полностью* лишить пользователя выбора. Ему часто придется принимать решения о том, как будет выглядеть его документ, как должна реагировать его веб-страница... -- о том, над чем он в конкретный момент работает. И здесь вы можете выплеснуть все буйство своей фантазии: нет ничего лучше, чем предоставить им в этих областях право принимать решения: и чем больше, тем веселее. Есть еще одна область, возможность выбирать в которой доставляет удовольствие: изменения внешнего вида, которые не приводят к изменениям в функциональности. Всем нравится поддержка скинов в WinAmp, каждый из нас меняет заставки рабочего стола. Поскольку подобные опции меняют внешний вид без изменения функциональности, поскольку пользователи могут без риска для себя проигнорировать эти опции и спокойно работать, использование подобных опции только приветствуется.

Руководство по UI дизайну для программистов
Глава4: Магическая сила метафор. Приглашения к действию.

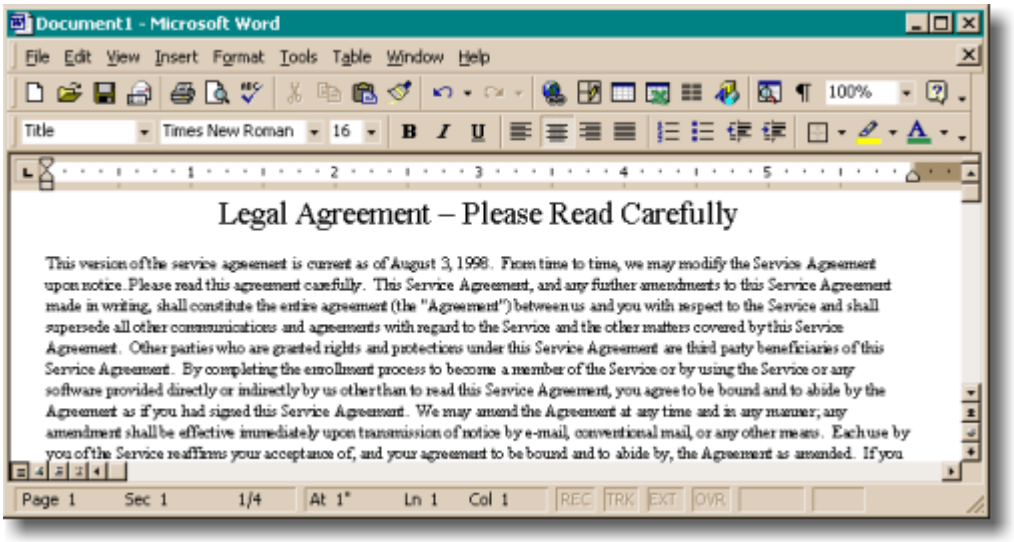
Разработать пользовательский интерфейс, в котором модель программы соответствует модели пользователя,-- задача не из легких. Иногда у пользователей *просто нет* конкретного представления о том, как работает программа и для чего она предназначена. В таком случае вам придется найти способ подсказать им, как функционирует ваша программа. В графических интерфейсах используется метод *метафор*. Метафоры, однако, работают не всегда, и важно понять, отчего это зависит, чтобы вы смогли научиться отличать плохую метафору от хорошей.

Самая известная метафора, применяемая и в *Windows* и в *Macintosh* -- это метафора "десктоп" (рабочий стол). Перед вами маленькие папочки с листочками-файлами внутри, последние можно перемещать из одной папки -- в другую. Метафора работает, потому что изображения папок напоминают реальные папки, которые мы используем для хранения и сортировки документов в своих кабинетах.

Перед вами изображение *Kai's Photo Soap*. Вы догадаетесь, как поближе рассмотреть картинку?

Очевидно, это не вызовет затруднений. Увеличительное стекло -- полновесная метафора из реального мира. Вы не опасаетесь, что приближение изменит размер картинки, потому что знаете, что увеличительное стекло сделать этого не может.

Метафора, пусть даже и не самая удачная, лучше чем ее отсутствие. Есть предположения о том, по какой иконке нужно щелкнуть, чтобы поближе рассмотреть документ в *Microsoft Word*?



Интерфейс *Word* содержит два маленьких увеличительных стекла, одно из которых (по непонятной причине) фигурирует под названием "Просмотр печати", второе же названо "Карта документа" (что это может значить, остается для меня загадкой). На самом деле, увеличить/уменьшить размер документа можно с помощью выпадающего списка, который на данный момент показывает "100%". Метафоры нет, и потому угадать, где скрывается функция приблизить/отдалить, сложнее. В нашем примере это не смертельно; подобная функция в текстовом редакторе не обладает такой степенью значимости, чтобы отвести под нее так много места на экране, как у *Kai*. Но я готов спорить, что этой функцией у *Kai* сумеет воспользоваться гораздо больше людей чем в *Word*.

Плохо подобранная метафора хуже чем ее отсутствие.

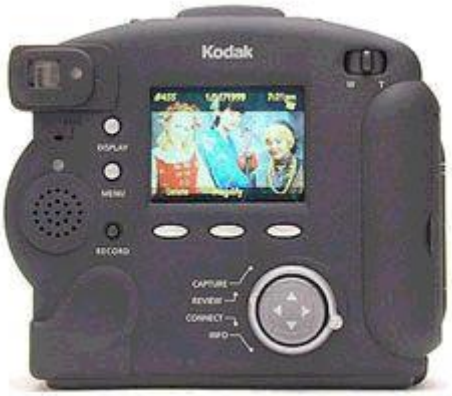
Вы помните иконку с портфелем в *Windows 95*? Симпатичная картинка размером с ноготь большого пальца красовалась на миллионах экранов в течение нескольких лет до тех пор, пока в *Microsoft* не сообразили, что она никому не нужна. А не нужна она была потому, что метафора хромала на все четыре ноги. Предполагалось, что это тот самый портфель, в который вы складываете документы, чтобы поработать с ними дома. Но если вы хотите взять с собой документы, вы же должны их сначала сохранить на дискету. Так вы складываете их в портфель или сохраняете на дискету? Я не знаю ответа на этот вопрос. Но метафору с портфелем я не понимаю. Я так и не сообразил, для чего она нужна.

Приглашения

Удачно разработанный дизайн хорош именно тем, что он позволяет сразу понять, как объект функционирует. На некоторых дверях на уровне руки расположены большие металлические пластины. Единственное, что вы можете проделать с ними -- толкнуть их. Как сказал Доналд Норман, они *приглашают* вас толкнуть их. На других дверях вы увидите большие закругленные ручки, которые просто вынуждают вас *потянуть* за них. Они заранее предполагают, как вам взяться за ручку, чтобы удобно открыть дверь. Ручка *приглашает* вас потянуть дверь на себя. Она заставляет вас *хотеть* сделать именно это.

Дизайн других объектов продуман не так хорошо и иногда бывает трудно предположить, для чего же они предназначены. Самый наглядный пример -- коробка для хранения CD, дизайн которой разработан так, что вы должны *особенным образом* расположить пальцы на ее крышке и потянуть в *определенном* направлении. Но ничто в дизайне не говорит о том, в каком же направлении тянуть или каким конкретно образом нужно положить пальцы. И если вы не знаете, как открыть коробку, вы затратите уйму времени и нервных клеток, чтобы понять, в чем заключается фокус, потому как никак иначе коробку не открыть.

Самый удачный способ создать "приглашение" -- вызвать ассоциацию с человеческой рукой в нужном месте объекта. Присмотритесь к (надо заметить, превосходной) цифровой камере Kodak DC-290 (вид спереди и сзади):



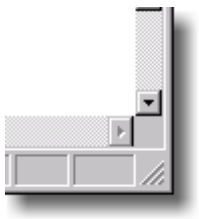
Спереди слева расположена большая резиновая накладка, которая выглядит так, как будто она специально приспособлена для пальцев вашей правой руки. Более того, сзади в нижнем левом углу находится выступ, чрезвычайно напоминающий отпечаток большого пальца. Если вы положите большой палец левой руки на этот выступ, указательный палец левой руки сам собой загнется и ляжет на фронтальную поверхность камеры, точно между объективом и еще одной резиновой накладкой. И вы ощутите необыкновенное чувство уюта, такое, какое не испытавали с тех пор, как сосали свой большой палец (в то время как указательный палец шаловливо ковырялся в носу).

Все, что хотят от вас дизайнеры *Kodak* (и в чем они очень успешны), -- чтобы вы обеими руками держали камеру -- в положении, которое обеспечивает наибольшую стабильность камеры и держит ваши пальцы подальше от объектива. Все эти резиновые накладки лишены функциональности, единственное их предназначение состоит в том, чтобы заставить вас держать камеру правильно.

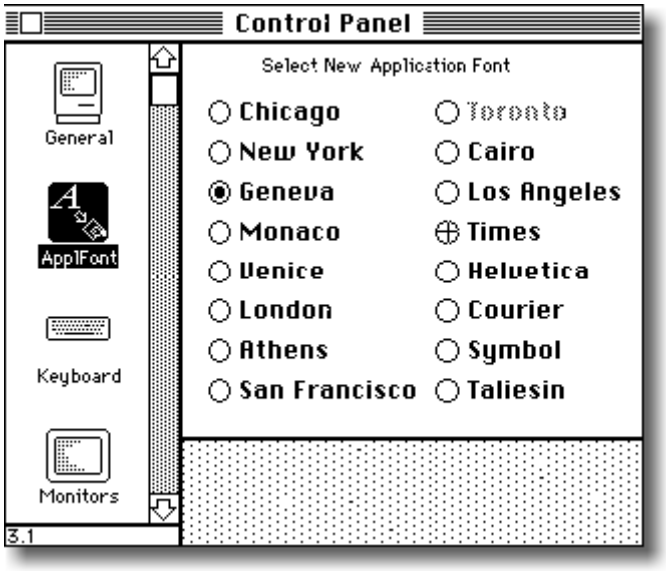
Точно также и хороший компьютерный интерфейс-дизайн применяет разного рода "приглашения". Где-то около десяти лет назад в моду вошли "трехмерные" кнопки: различные оттенки серого цвета, использованные в дизайне кнопок, придавали им выпуклый вид. Не для того, чтобы круто выглядеть: выпуклость кнопок *приглашала* к нажатию. Сам вид кнопок предполагал, что с ними нужно делать, а именно -- кликать по ним. К сожалению, многие веб-страницы сегодня (не осознавая ценности приглашений) предпочитают кнопки, которые просто выглядят круто. В результате, нам с вами иногда приходится поломать голову, чтобы сообразить, куда же здесь все-таки надо кликнуть. Взгляните на следующий баннер:

Кнопки "*Go*" и "*Login*" выглядет выпукло, так, что вы знаете, что по ним можно кликнуть. Кнопки "*Site Map*" и "*Help*" лишены подобной доброжелательности, более того, они в точности напоминают дизайн лже-кнопки "*Quotes:*", которая и вообще-то кнопкой не является.

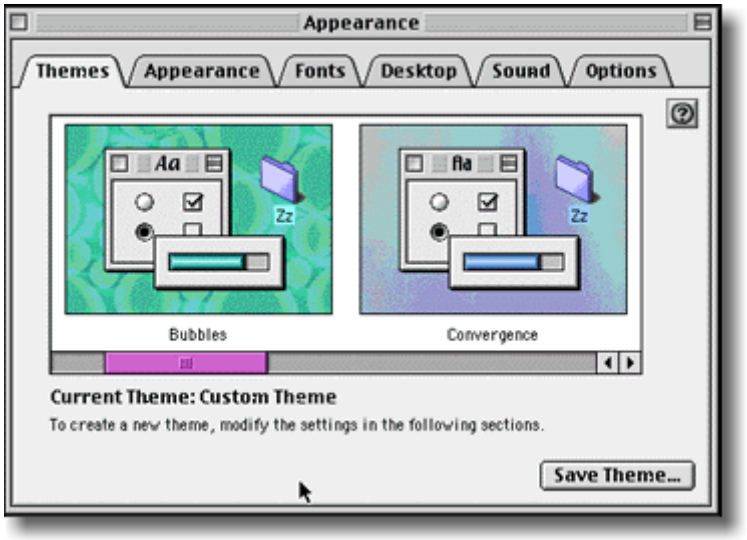
Около четырех лет назад многие приложения стали выходить с измененным дизайном окна: в нижнем правом углу появился выпуклый квадратик с тремя диагональными линиями. Он *приглашает* вас потянуть за него. Он просто-таки *умоляет* "потяни!", и окно растянется.



Ну и наконец, один из лучших примеров *приглашения*:*диалог с закладками (tabbed dialog)*. Помните, как выглядела старая добрая панель управления на Мак'е?



Идея состояла в следующем: вы выбираете одну из иконок из левого прокручивающегося списка. Щелчок мышью по выбранной иконке приводит к изменению правого списка. По какой-то невыясненной причине, подобный окольный путь казался дизайнерам интерфейса невероятно логичным, но многие пользователи просто опускали руки, не в состоянии понять, что же происходит. Среди прочего, для них так и осталось загадкой, как перемещаться по левому списку, чтобы увидеть все, а не только первые четыре иконки. Но, что более важно, большинство из них так и не уловили связи между иконками и самим диалогом. Иконки просто не были похожи на элементы панели управления. Начиная с 1992, подобные интерфейсы были постепенно вытеснены *диалогами с закладками*.



Приглашение -- в чистом виде. Из картинки *очевидно*, что у вас 6 закладок; *очевидно*, на какой из них вы сейчас находитесь; *очевидно*, как перейти на другую.

Когда *Microsoft* решил проверить удобство пользования таким интерфейсом, оказалось что оно выросло с 30% (старый дизайн *Macintosh*) до 100%. Каждый из тестируемых понял, как пользоваться интерфейсом. Осознавая небывалый успех этой метафоры и зная, что исходный ход панели с закладками встроен в *Windows* и доступен практически бесплатно, я не перестаю удивляться, почему до сих встречаются приложения, которые не используют все преимущества этого диалога. Подобные приложения страдают реальными, подчас тяжелыми заболеваниями, только потому что они лишают пользователя удобств общения с ними.

Основные программы пакета *Microsoft Office* -- *Word* и *Excel* -- разрабатывались с нуля программистами компании. Другие же были куплены на стороне: *FrontPage*, например, у *Vermeer*, или *Visio* -- у *Visio*. Что у этих программ общего? Дизайн обеих создавался с самого начала так, чтобы они выглядели и работали как приложения *Microsoft Office*.

Решение имитировать дизайн приложений *Microsoft Office* было принято не просто ради того, чтобы подлизаться к *Microsoft*. И даже не для того, чтобы впоследствии продать свою программу гиганту. На самом деле, создатель *FrontPage* Чарльз Фергюссон не скрывает своей неприязни к *Microsoft*, более того, он неоднократно призывал Департамент юстиции США принять хоть какие-нибудь меры по отнношению к *Редмонским бестиям* (до тех пор, пока не продал им свою компанию, после чего его положение сильно усложнилось). *Vermeer* и *Visio* скопировали пользовательский интерфейс *Microsoft Office*, должно быть, просто оттого, что это было выгодно: быстрее и проще, чем изобретать велосипед.

Когда менеджер отдела программирования *Microsoft* Майк Матье загрузил *FrontPage* с веб-страницы *Vermeer*, оказалось, что программа работает во многом так же как и *Word*. Поскольку она работала в соответствии с его ожиданиями, пользоваться ею было просто. И эта простота в использовании программы в одночасье произвела самое благоприятное впечатление на господина Матье.

Ну а если программа производит на *Microsoft* благоприятное впечатление, компания готова выложить за нее кругленькую сумму в районе $150 миллионов. Ваши цели, возможно, поскромнее: вы просите, наверное, что-то около $39 за то благоприятное впечатление, которое производит ваша программа. Но суть одна: постоянство дизайна программы есть причина удобства ее использования. Удобство в пользовании, в свою очередь, вызывает у человека приятные ощущения, которые для вас выражаются в росте суммы на банковском счету.

Дизайн элементов управления, выдержанный в едином ключе для различных программ, помогает пользователю обучиться работать с новой программой. До появления графических интерфйсов, каждому разработчику новой программы приходилось придумывать сами основы ее пользовательского интерфейса. Люди были вынуждены заучивать команды программ, с которыми они часто работали, чтобы быть в состоянии с ними работать. Самой нужной для пользователя (и необходимой для программы) была команда «выход»; многообразие вариантов ее исполнения поражает воображение. Приверженцы *Emacs* запоминали «:q!» (и ничего больше) на случай, если им не посчастливится застрять в текстовом редакторе *«vi»*, в то время как пользователи *«vi»* старались не забыть команду "C-x C-c" для выхода из *Emacs* (в *Emacs* даже придумали специальное графическое изображение для своих управляющих последовательностей).

В прекрасной стране *DOS* нечего было и думать запускать *WordPerfect*, пока к клавиатуре не был прикреплен кондовый пластиковый шаблон, который напоминал вам, какую команду выполняет комбинация Alt+Ctrl+F3. Я просто запомнил, что спасительная клавиша F7 вызволяет тебя оттуда.

Более того, небольшие расхождения в таких вещах, как набор команд на клавиатуре, заданный по умолчанию, могут просто свести вас с ума. Я настолько привык нажимать Ctrl+Z для отмены действия в приложениях *Word*, что постоянно минимизирую размер окна (Ctrl+Z), когда работаю в *Emacs*. (Самое смешное, что *Emacs* использует Ctrl+Z для минимизации окна как раз «для совместимости» с тем ужасным пользовательским интерфейсом csh -- *C shell* от *UNIX*.) Это как раз пример тех самых маленьких огорчений, которые в сумме дают общее чувство неудовлетворения.

И еще один пример: так же как и в *Emacs*, в *Pico* команда Ctrl+K применяется для удаления строк, но с *небольшими* особенностями в поведении, которые обычно калечат документ, с которым я в данный момент работаю в *Pico*. Уверен, что у вас наберется десяток собственных примеров.

На заре развития *Macintosh*, еще до рождения *Microsoft Windows*, проповедники *Apple* утверждали, что средний пользователь *Macintosh* использовал в своей работе больше программ, чем средний пользователь *DOS*. Я не помню точных цифр, но речь шла об одной -- двух программах для пользователя *DOS* и двенадцати -- для пользователя *Macintosh*. И все потому, что обучиться новой программе *Macintosh* не составляло никакого труда -- от того, что все они работали приблизительно одинаковым образом.

Постоянство в дизайне -- фундаментальный принцип хорошего *UI* дизайна, хотя он и является просто следствием аксиомы «модель программы должна соответствовать модели пользователя», на основании того, что модель пользователя отражает предыдущий опыт пользователя. Если он научился тому, что двойной щелчок мышью в тексте приводит к выбору слова, то при работе с новой программой он сразу догадается, что для того, чтобы выбрать слово, он должен дважды щелкнуть по нему мышью. И -- будьте бдительны! -- двойной щелчок по слову должен приводить к выбору слова, иначе у вас возникает проблема с удобством пользования программой.

Но если постоянство в дизайне -- настолько очевидно полезная вещь, так ради чего я тут трачу ваше и свое время на пропаганду? К несчастью, в этом мире существует темная сила, которая неутомимо борется с феей постоянства, и название ей -- естественная потребность дизайнеров и программистов в творчестве.

Мне ужасно не хочется быть тем, кто вам скажет «перестаньте творить», но если вы хотите создать удобный для пользователя интерфейс, вам, к сожалению, придется направить свои творческие порывы в иное русло.

Прежде чем начать работу над дизайном интерфейса, необходимо узнать, как работают другие распространенные программы. И затем -- скопировать их поведение как можно более точно. Если вы создаете текстовый редактор, позаботьтесь о том, чтобы ваша программа была похожа на Microsoft Word, вплоть до сочетаний клавиш в списке меню, которые есть и в вашей программе, и в Word. Некоторые из потенциальных пользователей вашей программы привыкли использовать комбинацию Ctrl+S для сохранения документа, другие -- Alt+F,S, третьи все еще пользуются Alt,F,S (отпуская клавишу Alt). Четвертые будут искать пиктограмму дискетки в верхнем левом углу. В ваших же интересах -- обеспечить функциональность всех четырех вариантов, иначе ваши пользователи не получат то, что они хотели.

Мне встречались компании, менеджеры которых гордятся тем, что намеренно создают программы, которые отличаются от *Microsoft*. Размахивая флагом «То, что это делает *Microsoft*, не значит, что это правильно», они создают пользовательские интерфейсы, неоправданно отличающиеся от того, к которому люди привыкли. Прежде чем и вы примкнете к их рядам, подумайте, пожалуйста, о следующем:

1. Пусть это и неправильно, но если *Microsoft* использует это в таких известных приложениях как *Word*, *Excel*, или *Internet Explorer*, миллионы людей будут думать, что это правильно, или, по крайней мере, является стандартом. Они изначально будут считать, что ваша программа работает подобным образом. Даже если вы думаете (как создатели *Netscape 6*, например), что Alt+Left -- не самая удачная комбинация для команды «назад», в этом мире существуют в буквальном смысле миллионы людей, которые попытаются ее применить. И если вы откажете им в этой малости -- исключительно из религиозной веры в то, что Билл Гейтс -- посланник дьявола, вы тем самым пожертвуете успехом собственной программы ради того, чтобы почувствовать себя спасителем человечества. Но благодарностей пользователей вы не получите.
2. А отчего вы так уверены, что это неправильно? *Microsoft* тратит больше денег на тестирование *usability* чем вы. Они ведут детальную статистику данных, полученных на основе анализа миллионов звонков в службу технической поддержки. И -- ставлю сто против одного -- они сделали это так, потому что больше людей могут сообразить, каким образом это функционирует.

Если вы хотите создать хорошую программу с удобным пользовательским интерфесом, оставьте свою религию за порогом офиса. Благодарю вас.

Microsoft не является единственным примером для подражания; если вы собираетесь открыть книжный интернет-магазин, вам придется позаботиться о том, чтобы ваша страница хотя бы семантически напоминала *Amazon*. На *Amazon.com* корзинка с покупками покупателя сохраняется в течение 90 дней. В вашу умную голову может прийти идея опустошать ее по истечении 24 часов.

Клиенты *Amazon*, побродив по вашему магазину, сложат покупки в корзину и вернутся через пару недель, ожидая увидеть свои еще не купленные продукты в корзине. Увидев же пустую корзину, они развернутся и больше не придут. Вы потеряли покупателя.

Если вы работаете над созданием профессионального графического редактора для дизайнеров, смею вас заверить, 90 процентов ваших потенциальных клиентов уже знакомы с *Adobe Photoshop*, так что постарайтесь сделать программу, работающую сходным с *Photoshop* образом в тех областях, где обе программы предлагают одинаковые функции. Если вы этого не сделаете, вам придется выслушивать обвинения пользователей в том, что с вашей программой трудно и неудобно работать (несмотря на то что вы-то знаете, что работать с ней легко). И все потому, что программа работает не так, как они того ожидают.

Более того, мы никак не можем отвыкнуть заново придумывать стандартные элементы управления вместо тех, что пришли к нам с *Windows*. О *Netscape 6* я даже не хочу говорить. Раньше можно было с уверенностью сказать, какая из программ была собрана с помощью C++ компилятора от Борланда: все они были декорированы большими жирными кнопками *OK* с гигантскими зелеными галочками. Но даже это выглядело не так ужасно, как этот элемент из *Kai's Photo Soap*:

Класс, просто дух захватывает от великолепия дизайна, но мне отчего-то большое перечеркнутое «Ø» напоминает *«OK»*, хотя и значит оно *«нет»*. К тому же, согласно стандарту *Windows* кнопка *«OK»* находится слева, и поэтому я не перестаю лупить по неправильной кнопке. Единственная польза от всех подобных забавных символов на месте кнопок *«OK»* и *«Отмена»* заключается в том, что они демонстрируют ваш творческий потенциал (я считаю, что есть еще один аспект -- такие диалоги проще *локализовать* -- А.М.). И если творческий потенциал *Kai* заставляет людей делать ошибки, что ж, это цена, которую они должны платить за привилегию наслаждаться результатами работы художника. (Другая проблема этого, с позволения сказать, диалога в том, что вы не можете передвигать его по экрану: он лишен стандартного заголовка. То есть, если он открывается в том месте экрана, где находится информация, котрая вам нужна, чтобы ответить на вопрос в окошке диалога, будем считать, вам просто не повезло.) Теперь и вы обладаете полезной информацией о преимуществах оригинального художественного графического дизайна (как у *Kai*): он радует глаз и привлекает клиентов. Фокус заключается в том, чтобы добиться этого без нарушения правил. Изменить внешний вид окна диалога можно -- немного, но не за счет изменения функциональности.

Первая версия Juno была оснащена стандартным диалоговым окном регистрации, с именем пользователя и паролем. После набора имени нужно было нажать клавишу TAB для перехода на поле пароля. Такой стандартный для *Windows* подход оказался неприемлемым для одного из старших программистов *Juno*, который привык работать с *UNIX*, где для перехода с поля имени на поле пароля нужно нажать *ENTER* (а не *TAB*). Логика подсказывает, что программист *UNIX* не является идеальным примером типичного пользователя при создании программы для целевой группы неопытных пользователей *Windows*. Но наш старший программист проявил беспримерную настойчивость в деле изменения стандарта *Windows*. «То, что это делает *Microsoft*, не значит, что это правильно»,-- приговаривал он.

Поэтому программистам пришлось потратить немалое количество времени на сооружение невероятного по своей сложности диалогового окна, которое должно было работать в обход стандартных установок *Windows*. (Плыть против течения всегда сложнее, и требует гораздо больших затрат.) Этот код стал кошмарным сном для команды поддержки; с большими трудностями он был портирован при переходе с *Win16* на *Win32*. Он делал вовсе не то, что ожидали пользователи. И когда к команде разработчиков присоединились новые программисты, они так и не смогли понять причину появления в коде странного подкласса для диалогов.

Несметные отряды программистов пытались поменять стандартные элементы управления *Windows*: кнопки и полосы прокрутки, панель инструментов и панель меню (с которой разработчики *Microsoft Office* особенно любят экспериментировать). В *Netscape 6* изменены вообще все стандартные элементы управления *Windows*. Эксперименты эти часто давали непредсказуемый результат. Самый наглядный пример -- поле для ввода текста.

Если вы напишите свой собственный вариант, огромное количество утилит, о существовании которых вы просто не подозреваете (например, поддержка китайского языка или набора текста справа налево), не будут работать, так как они не в состоянии распознать нестандартный элемент управления. При тестировании *бета*-версии *Netscape 6* было замечено, что поле ввода *URL*, снабженное нестандартной версией управления, не поддерживает обычные функции управления, такие как щелчок правой кнопкой мыши для вызова контекстного меню.

Если вам когда-нибудь придется спорить о постоянстве в исполнении дизайна различных программ с ярым противником *Microsoft* или с каким-нибудь креативным дизайнером, вы непременно услышите цитату из Эмерсона: «Постоянство есть признак скудости ума». Заметьте, цитата неверна. В оригинале она звучит так: «Бессмысленное постоянство есть признак скудости ума» ('A foolish consistency is the hobgoblin of little minds.' Emerson)

Хорошие дизайнеры пользовательских интерфейсов подходят к вопросу о постоянстве с умом; возможно, им не часто выдается возможность продемонстрировать свой творческий потенциал, но в конечном итоге, это лишь на пользу их клиентам.

Руководство по UI дизайну для программистов
Глава6: Как создать дизайн для людей, которым есть чем заняться в этой жизни

Когда вы работаете над созданием пользовательских интерфейсов, полезно помнить о двух принципах:

1. У пользователей нет документации, а если бы она и была, они бы ее не читали.
2. На самом деле, пользователи не умеют читать, а если бы и умели, то не не стали бы.

По правде говоря, это не фактические данные. Но вам следует действовать так, как если бы они были таковыми, ибо это поможет сделать вашу программу проще и удобнее в использовании. Учитывать эти принципы в работе над дизайном интерфейса значит отдавать дань уважения пользователю, что, в свою очередь, значит не слишком уважительно относиться к его способностям. Звучит противоречиво? Позвольте объясниться.

Что означает «сделать простым в использовании»? Чтобы измерить простоту использования пограммы, можно представить, какой процент обычных пользователей сможет выполнить ряд заданий в отведенное время. Предположим, что ваша программа позволяет конвертировать фотографии, сделанные цифровой камерой, в веб-фотоальбом. Если вы посадите группу среднестатистических пользователей выполнять эту задачу, окажется, что чем удобнее, чем проще ваша программа в использовании, тем выше процент людей, которые успешно справяться с созданием веб-фотоальбома. Подойдем к вопросу более научно. Представьте себе 100 пользователей. Не обязательно,что они разбираются в компьютерах. Они наделены различными талантами, но некоторые из них совершенно точно не обладают талантами в компьютерной области. Некоторых из них постоянно отвлекают, когда они работают над выполнением вашего задания. Звонит телефон. *ЧТО?* Ребенок плачет. *ЧТО?* Котенок прыгает по столу, пытаясь поймать мышь. *Я НЕ СЛЫШУ ВАС!*

Даже не доводя эксперимент до конца, ясно, что некоторые пользователи просто завалят выполнение задания, или же затратят на него невероятное количество времени. Нет, я не собираюсь сказать «по собственной глупости». Напротив, многие из них достигли успеха: одни -- в молекулярной физике, другие -- в тройном прыжке, но в ситуации один на один с вашей программой они просто не используют на полную катушку свою моторику или серое вещество. Ваша программа получает, в лучшем случае, 30 тридцать процентов их внимания. Это значит, что вам придется иметь дело с пользователем, который при работе с компьютером полностью не выкладывается.

Пользователи не читают руководств.

Во-первых, руководства у них просто нет. Его может не существовать в природе. Если же оно существует, не значит, что оно находится под рукой у нашего пользователя: он летит в самолете, или пользуется скачанной с вашего веб-сайта демо-версией программы, или он сидит дома, а руководство пылится на работе, или же начальник отдела забыл отдать ему руководство, и т.д. и т.п. Даже если у него есть руководство, положа руку на сердце, заявляю: он его не откроет, а если и откроет, то только тогда, когда у него не останется никакого другого варианта. За очень редким исключением, люди не будут уютно устраиваться в любимом кресле под зеленым торшером, чтобы прочесть руководство по эксплуатации прежде, чем начать пользоваться новой для них программой. В общем и целом, ваши клиенты работают над решением какой-либо задачи, и для них чтение руководства представляется лишь ненужной тратой времени, или же преградой, которая отвлекает от достижения цели.

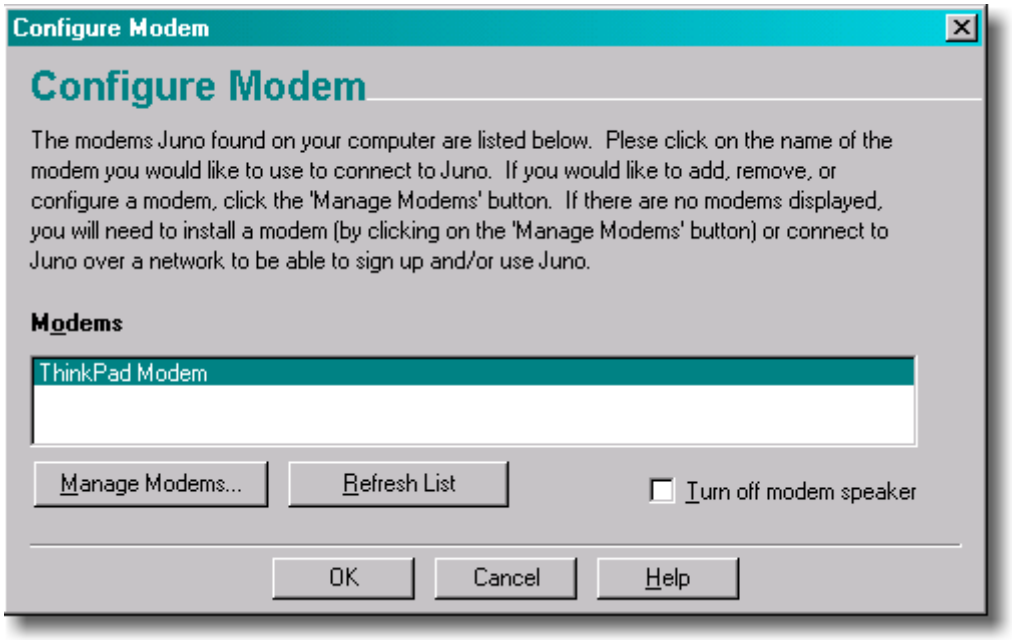
Тот факт, что вы читаете эту книгу является доказательством вашей принадлежности к элитной группе высокограмотных людей. Да-да, мне известно, что люди, пользующиеся компьютером, более или менее умеют читать, но поверьте, для большинства из них чтение -- лишь помеха. Язык, на котором написано руководство, не обязательно является их родным языком и они могут не знать его в совершенстве. Или окажется, что вашей программой пользуется ребенок! Да, они смогут расшифровать то, что написано в руководстве -- при условии, что это необходимо, но точно не будут его читать, если без этого можно обойтись. Пользователи читают руководства исключительно по принципу «до зарезу нужно, но спросить некого».

Вывод из всего вышеизложенного краток: у вас нет иной альтернативы, чем кроме как спроектировать программу так, чтобы потребность в чтении руководства не возникала. Единственное исключение, которое мне приходит на ум, -- когда у ваших клиентов нет никаких базовых знаний, то есть они не понимают, что, собственно говоря, эта программа делает, но осознают, что это знание им необходимо. Хорошей иллюстрацией этого исключения является невероятно популярная программа для бухгалтерского учета в малом бизнесе - *QuickBooks* от *Intuit*. Большинство из тех, кто этой программой пользуется -- представители малого бизнеса, которые не имеют ни малейшего представления о том, что такое бухучет. Авторы руководства к *QuickBooks* знали об этом, как и о том, что им нужно обучить людей основым принципам бухчета. Без руководства пользоваться *QuickBooks* было бы невозможно. С другой стороны, люди, знакомые с бухучетом, могут спокойно работать с программой без руководства.

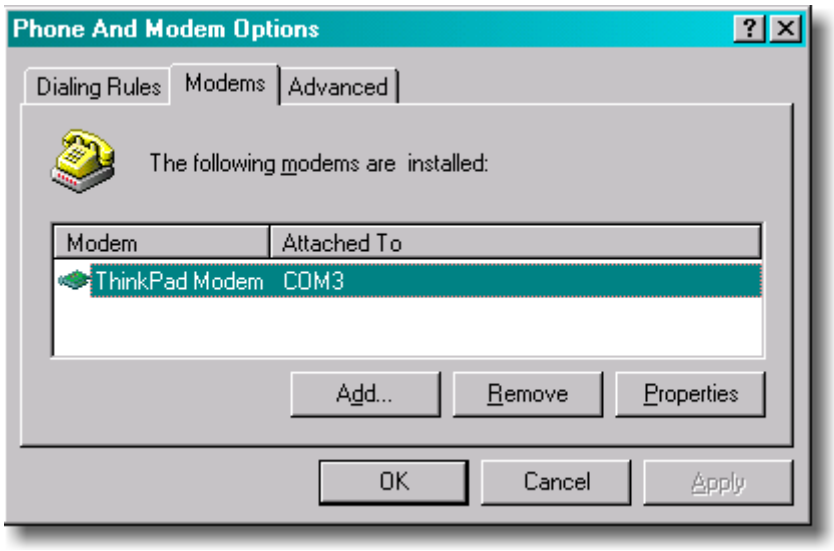
На самом деле, пользователи не читают *ничего*. Даже комиксы.

Возможно, это звучит жестко, но в этом легко убедиться, проведя *usability* тестирование. Вы увидите, что немалое количество людей просто не читают слова, которые вы вывели на экран. Когда на экране появляется окно с предупреждением об ошибке, его просто не читают. Это может стать для вас -- программиста -- разочарованием, потому что вам кажется, что вы ведете с пользователем диалог. «Товарищ пользователь! Этот документ Вы открыть не можете, потому что программа не поддерживает его формат!» Но опыт показывает, что чем больше слов в диалоговом окне, тем меньшее число пользователей их читают.

Тот факт, что пользователи не читают руководств, заставляет дизайнеров программ предположить, что они должны чему-то обучить своих пользователей. Обучение выливается в описание различных вещей в процессе того, как работает программа. В принципе, само по себе это неплохо, но в действительности это оборачивается против вас -- по причине стойкого отвращения людей к излишнему чтению. Опытные дизайнеры пользовательских интерфейсов в буквальном смысле сводят количество слов на экране к минимуму, чтобы увеличить вероятность того, что они будут прочитаны. Во времена моего участия в проекте *Juno*, *UI* программисты в конторе осознавали важность этого принципа и старались писать короткие, ясные, простые тексты. Грустно об этом говорить, но исполнительным директором компании в то время был специалист по английской филологии с дипломом элитарного колледжа в кармане; никаких курсов по дизайну пользовательских интерфейсов он не заканчивал, зато, должно быть, считал себя хорошим литературным редактором. Посему на скупую прозу профессиональных дизайнеров он наложил запрет и заменил ее своими графоманскими изысками. Типичное диалоговое окно *Juno* выглядит теперь следующим образом:



Аналогичное же диалоговое окно в *Windows* выглядит так:

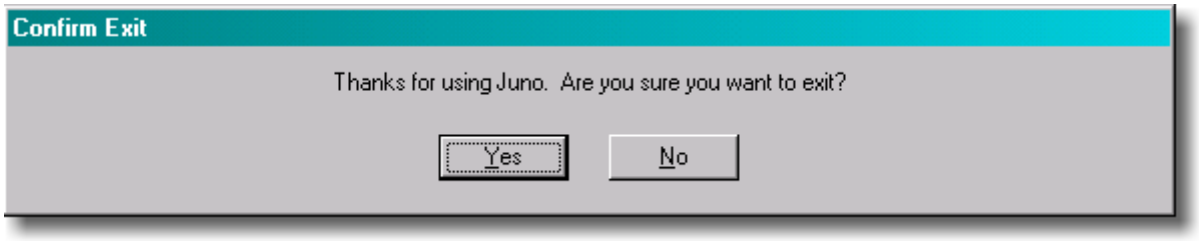


Интуитивно можно подумать, что версия *Juno* с 80 словами лучше (т.е. проще в использовании), чем состоящая из пяти слов инструкция *Windows*. На самом деле, при проведении *usability*-тестирования подобных вещей можно обнаружить, что:

- продвинутые пользователи подобные инструкции пропускают. Они знают, что надо делать, и времени на чтение сложных инструкций не имеют;
- большинство неопытных пользователей подобные инструкции пропускают. Они не любят читать и рассчитывают на то, что установки по умолчанию им подойдут;
- оставшиеся неопытные пользователи честно стараются прочитать инструкции (некоторые из них просто потому, что участвуют в тестировании и считают себя обязанными выполнять все инструкции). Но их часто приводит в затруднение само количество слов и понятий. Даже если при первом появлении диалогового окна они были достаточно уверены в том, что знают, что с ним делать, многословные инструкции приводят их в большее затруднение.

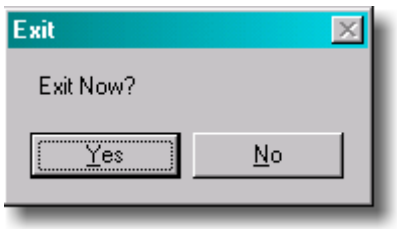
Итак, очевидно, что менеджмент *Juno* перестарался вне всякой меры. Позволю себе еще одно общее замечание: если вы заканчивали филологический факультет университета, помните о том, что вы принадлежите к иной лиге грамотности нежели среднестатистический пользователь, и поэтому вам следует особенно осторожно подходить к проектированию содержания диалоговых окон. Делайте тексты как можно более краткими, простыми, доступными, избавляйтесь от придаточных предложений и деепричастных оборотов. Ни в коем случае не пишите текстов, похожих на заметки, размещенные на доске объявлений филфака в вашей альма-матер. Даже простое слово «пожалуйста» -- атрибут вежливости в обычной жизни -- в диалоговом окне лишь замедляет работу: перенасыщенный словами текст значительно уменьшает количество людей, которые его прочтут.

Другой важный момент: **многие испытывают страх перед компьютером**. Вам, наверное, это известно, не так ли? Но вы, скорее всего, не осознаете всех последствий этого страха. Однажды я видел, как одна моя хорошая знакомая пыталась выйти из *Juno*. По какой-то причине она испытывала при этом большие затруднения. Я заметил, что выход из *Juno* предваряется диалоговым окном со следующим содержанием:



Моя знакомая судорожно нажимала на кнопку «Нет», и немало удивлялась тому, что программа не закрывалась. Сам факт, что программа ставила под сомнение ее выбор, заставил ее немедленно прийти к заключению, что она делала что-то неправильно. Обычно программа просит подтвердить команду тогда, когда вы можете совершить что-то, о чем в последствии пожалеете. Поэтому моя знакомая предположила, что если компьютер ставит под сомнение правильность ее решения, то он прав, потому что компьютер -- это компьютер, а она всего лишь человек. И она выбрала «Нет».

Неужели так сложно прочитать несчастные одиннадцать слов? Очевидно, да. Во-первых, выход из *Juno* не влечет за собой никаких разрушительных последствий, поэтому просьба о подтверждении команды является излишней (примером тому являются все существующие *GUI* программы, которые обходятся без подтверждения). Если же вы убеждены в том, что подтверждение о выходе жизненно необходимо для вашей программы, эту просьбу можно выразить не в одиннадцати, а в двух словах: «Выйти сейчас?».



Диалог, лишенный абсолютно бесполезного «Спасибо» и вызывающей сожаление фразы «Вы уверены?», повлечет за собой гораздо меньше проблем. Человек гарантированно прочтет два слова, вздохнет, и нажмет кнопку «Да».

Да, конечно, диалоговое окно *Juno* с просьбой о подтверждении команды «выход» может вызвать проблемы у небольшого количества людей, скажете вы. Неужели это так важно?

Рано или поздно, но все пользователи справятся нажать кнопку «Да» и благополучно закончить программу. Но именно в этом и заключается разница между программой, пригодной к использованию, и программой, удобной в использовании. Даже самые умные, опытные, самые продвинутые пользователи оценят то, что вы делаете для рассеянных, неопытных начинающих пользователей. Ванны в номерах хороших гостиниц оснащены специальными большими ручками, с помощью которых можно легко выбраться из ванны. Изначально они были созданы для инвалидов, но ими пользуются все. Они делают жизнь -- в том числе и физически здоровых людей -- немного проще.

В следующей главе мы немножко побеседуем о компьютерной мыши. Многие пользователи не любят/не желают/не умеют читать. Точно также многие не слишком хорошо знают, как нужно пользоваться мышью, поэтому вам нужно им в этом помочь.

Руководство по UI дизайну для программистов
Глава7: Как создать дизайн для людей, которым есть, чем заняться в этой жизни. Часть 2.

Когда компания *Microsoft* еще только делала свои первые шаги, Bruce "Tog" Tognazzini вел в журнале для разработчиков *Apple* колонку о дизайне интерфейсов. В ней обсуждались многие интересные проблемы *UI* дизайна. Колонка живет и по сей день: на его личной веб-странице, а также -- в скомпанованном виде -- в блистательных книгах Тога. Одна из них -- «Tog on Software Design» -- увлекательное и полезное введение в науку *UI* дизайна. (Собрание «Tog on Interface» еще лучше, но к сожалению, в печатном виде не издавалось.)

Тог придумал концепцию «панель меню высотой в километр» для того, чтобы объяснить преимущество панели меню *Macintosh* перед той же панелью в *Windows*. В *Windows* панель появляется внутри каждого нового окна приложения. Чтобы открыть пункт меню (например, *File*), нужно прицелиться и поймать мышью прямоугольничек размером полтора сантиметра на сантиметр. От пользователя требуется аккуратное и точное позиционирование стрелки мыши как по вертикали, так и по горизонтали.

В *Macintosh* же панель приклеена вверху видимой области экрана. Это дает вам возможность швырять мышь как угодно высоко по вертикали -- курсор всегда окажется вверху экрана, то есть точно на высоте панели меню. Как следствие, прицел ваш точно также ограничен полутора сантиметрами в ширину, но уже километром в высоту. Все, что вам нужно сделать, это лишь правильно расположить курсор по горизонтали; о позиционировании курсора по вертикали уже позаботился *Macintosh*, и открыть пункт меню стало вдвое проще.

Этот же принцип лежит и в основе проводимой Тогом мини-викторины: какие пять областей экрана легче всего найти с помощью мыши? Ответ: четыре угла экрана (вы можете в буквальном смысле швырнуть мышь, совершенно не заботясь о позиционировании стрелки) и текущая позиция курсора.

Принцип панели высотой в километр достаточно известен, но, похоже, не совсем очевиден, поскольку команда разработчиков *Windows 95* ухитрилась его совершенно неверно интерпретировать. Кнопку «Старт» они расположили в самом нижнем левом углу экрана, но -- почти в самом нижнем левом углу экрана. Отступ снизу составляет два пикселя, отступ слева -- тоже два пикселя. Из-за какой-то несчастной пары пикселей *Microsoft* «вырвал поражение из клыков победы», как образно выразился Тог, и вынудил пользователей терять больше времени на то, чтобы кликнуть по кнопке «Старт». Кнопка могла бы быть квадратным километром, абсолютно тривиально достижимым мышью. Но ради чего-то -- мне не известного -- этим пренебрегли.

В прошлой главе мы выяснили, что пользователи очень не любят читать и снисходят до этого только тогда, когда им совершенно непонятно, как выполнить поставленную задачу. Следующая мысль аналогична:

Пользователи не в состоянии идеально контролировать движения мыши.

Не поймите меня буквально. Я имею в виду, что ваша программа должна быть разработана таким образом, чтобы от пользователя не требовалось мастерское владение техникой позиционирования мыши. И вот почему:

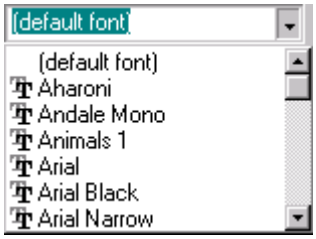
1. Иногда приходится пользоваться не самыми оптимальными манипуляторами -- трэкболами, трэкпадами, или, если совсем не повезло, крошечными красненькими пимпочками на *ThinkPad* ах, -- которые сложнее контролировать, чем обычную мышь.
2. Иногда приходится пользоваться мышью не в самых благоприятных условиях: стол завален бумагами, мышь давно не чищена, или сама мышь -- не мышь, а клон ее, купленный за 50 рублей на рынке, который находится с курсором в натянутых отношениях.
3. Иногда человек, сидящий за компьютером,-- новичок, моторные навыки которого еще не достаточно развиты для совершенного владения мышью.
4. Некоторые люди никогда не смогут развить эти самые навыки, по разным причинам: с вашей программой будут работать в том числе и люди, страдающие от артрита, тремора, кистевого туннельного синдрома, инвалиды, маленькие дети или старики.
5. Многим просто не удается дважды щелкнуть мышью, не передвинув ее при этом на пару сантиметров. В результате они часто таскают окна по экрану, хотя на самом деле просто хотели запустить приложение. Экраны их компьютеров похожи на свалку мусора, потому что в пяти случаях из десяти попытка запустить приложение заканчивается перемещением ее иконки на экране.
6. Некоторые люди считают, что постоянное применение мыши замедляет работу. Если вы заставите их выполнять многошаговую операцию с помощью мыши, они могут почувствовать, что темп их работы снизился, и решить в итоге, что созданная вами программа медленно реагирует. К чему это приводит? Надеюсь, что вы уже знаете ответ на вопрос. Да, к чувству неудовлетворения.

В пору моей работы над *Excel*, ноутбуки еще не были оснащены встроенными манипуляторами. Поэтому в *Microsoft* придумали специальный трэкбол, прикрепляющийся к боковой стороне клавиатуры. Современные устройства позволяют контролировать мышь с помощью кисти и пальцев. Это во многом похоже на письмо, моторные навыки которого развивают еще в школе. Но трэкбол контролируется только большим пальцем руки, что приводит к меньшей степени контроля. Многим удается контролировать позиционирование курсора с помощью мыши с точностью до одного -- двух пикселей, при работе же с трэкболом точность достигает лишь трех -- четырех.

Поэтому, когда я работал в команде *Excel*, я всегда настаивал на том, чтобы разработчики тестировали свои программы с использованием трэкбола -- для того, чтобы они могли оказаться в ситуации людей, которые не могут позиционировать курсор точно в том месте, где это нужно.

Один из элементов графического дизайна, который вызывает у меня наибольшее раздражение, называется выпадающий список. Выглядит он следующим образом:

Когда вы щелкаете по черной стрелочке, список раскрывается:

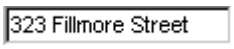


А теперь подумайте, сколько точно выверенных манипуляций мышью вам понадобится, чтобы выбрать *Times New Roman*, например. Сначала нужно щелкнуть мыщью по стрелочке, указывающей вниз. Затем, аккуратно перемещая квадратик управления полосы прокрутки, просматривать список до тех пор, пока не появится нужный шрифт. Многие подобные выпадающие списки бездумно разработаны таким образом, что показывают лишь два или три элемента списка. Поэтому прокручивание превращается в довольно кропотливую процедуру, особенно когда в списке десятки шрифтов. Вариантов немного: нужно либо осторожно тащить бегунок вниз по полосе прокрутки, либо постоянно щелкать мышью по нижней стрелке. Или же пытаться щелкать в пространстве между бегунком и нижней стрелкой -- что под конец, когда бегунок уже почти добрался до нижней стрелки, перестает работать. Раздражение в вас нарастает. Если вам удалось добраться до *Times New Roman*, нужно щелкнуть еще и по нему. Если вы промахнулись, придется начать сначала. А теперь умножьте результат на десять, если вам захотелось, чтобы каждая новая глава начиналась щеголеватой заглавной буквой, к примеру. Что вы чувствуете? Вы уже по-настоящему раздражены.

Крохотные выпадающие списки раздражают еще больше, потому что решение проблемы очевидно: выпадающий список нужно просто сделать таким длинным, чтобы он вмещал все элементы. 90 процентов же списков не задействуют все имеющееся пространство до низа экрана, что является попросту грехом. Если места между самим элементом управления и низом экрана недостаточно, чтобы вместить все элементы списка, надо использовать пространство до верха экрана, даже если список грозит занять все пространство от верхней до нижней границы физического экрана. В случае, когда у вас больше элементов, чем список в высоту экрана может вместить, установите на нем автоматическую прокрутку. Это намного лучше, чем заставлять ни в чем неповинного пользователя возиться с уворачивающимся от мыши списком прокрутки.

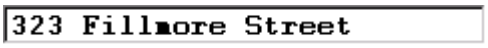
Более того, не заставляйте меня щелкать по этой крошечной стрелочке справа для того, чтобы открыть список. Позвольте мне щелкнуть в любом месте этого окна. Это увеличивает цель раз в десять и облегчает позиционирование стрелки в окне.

Обратимся к еще одной проблемной зоне «мышеведения»: поля для редактирования текста (*edit boxes*). Возможно, вы заметили, что в любом поле редактирования на *Macintosh* используется широкий, жирный, крупный шрифт под названием *Chicago*. Выглядит он несколько грубовато и безмерно раздражает дизанейров. Обычных дизайнеров (не путать с дизайнерами графических интерфейсов!) учат тому, что *пропорциональные* шрифты более элегантны, изящны, и, к тому же, легче читаются. Что верно, то верно. Только все это применимо к бумаге, а не к экрану. При редактировании текста преимущество получают моноширинный шрифты: такие узкие буквы как «l» и «i» в них более различимы, и их легче выбрать. Этот урок мне преподал шестидесятилетний человек, который в процессе *usability*-тестирования мучительно пытался отредактировать название своей улицы, что-то типа *Fillmore Street*. Мы использовали тогда шрифт *Arial 8*, и поле для редактирования выглядело так:



Обратите внимание, ширина букв «L» и «I» составляет буквально один пиксел. Разница между маленькими «l» и «i» тоже в один пиксел. (*Аналогично*: уловить разницу между прописными «RN» и «M» почти невозможно, поэтому в поле редактирования могло быть написано и *Fillrnore*.)

Очень немногие могут заметить, что они неправильно напечатали *Flilmore*, *Fiilmore* или *Fillrnore*. Заметив, они потратят прорву времени на то, чтобы с помощью мыши выбрать неверно напечатанную букву и исправить ее. Проблемы возникнут и с мигающим курсором, ширина которого два пикселя, если нужно выбрать одну букву. А теперь посмотрите, насколько легче это сделать, если используется крупный шрифт (в данном случае *Courier Bold*):



Да, но он занимает больше места и не выглядит стильно, скажут ваши дизайнеры. Убедите их! Подобные шрифты удобнее использовать, и это чувствуется: при наборе текста вы видите, как он приобретает форму, объем и ясность, и его гораздо легче редактировать.

Вот вам пример логики программиста: существуют только три цифры: *0*, *1*, и *n*. Если *n* разрешено, то все *n* одинаковы. Это логическое заключение есть следствие программистской веры (возможно, оправданной) в то, что код не должен содержать никаких числовых постоянных за исключением *0* и *1*. (Любые другие постоянные называются «магическими числами», и у меня нет ни малейшего желания углубляться в эти дебри.)

Поэтому программисты склонны думать, например, что если приложение позволяет открывать более чем один документ, то оно должно позволять открывать бесконечное количество документов (в пределах памяти процессора), или, по меньшей мере, *два в тридцать второй степени* документа (единственное магическое число, которое программист способен допустить). Программист с презрением отнесется к программе, которая ограничит число открытых документов до *20*. Что такое *20*? Почему *20*? *20* нельзя получить возведением двойки в степень!

Следствием аксиомы «все *n* равнозначно вероятны» явилось еще и то, что программисты были склонны считать, что если пользователям позволено изменять размер окон и перемещать их, то они должны получить безоговорочное право перемещать окна по всей поверхности экрана, вплоть до последнего пикселя. В конце концов, позиционирование окна в двух пикселях от верхней границы экрана также «равнозначно вероятно», как и позиционирование окна точно у верхней границы экрана.

Но вот это-то и неверно. Как оказалось, существует много веских причин для того, чтобы вы захотели разместить окно точно у верхней границы экрана (например, потому что это максимизирует рабочую поверхность экрана). И ни одной причины для того, чтобы оставить два пикселя свободного места между верхней границей экрана и верхней границей окна. Получается, что *0* гораздо более вероятен чем *2*.

Программисты в *Nullsoft*, создатели WinAmp, каким-то образом смогли избежать этой ловушки программистского мышления, в которой все остальные просидели десятилетие. У *WinAmp* есть замечательная функция. Когда вы, перемещая окно, оказываетесь на расстоянии нескольких пикселей от границы экрана, окно автоматически приклеивается к границе экрана. А это, скорее всего, как раз то, что вам было нужно, поскольку *0 гораздо более вероятен чем 2*. (У главного окна *Juno* есть подобная функциональность: это единственное приложение из всех мною виденных, которое «заперто в прямоугольник» на экране так, что его нельзя перетащить за границу прямоугольника.)

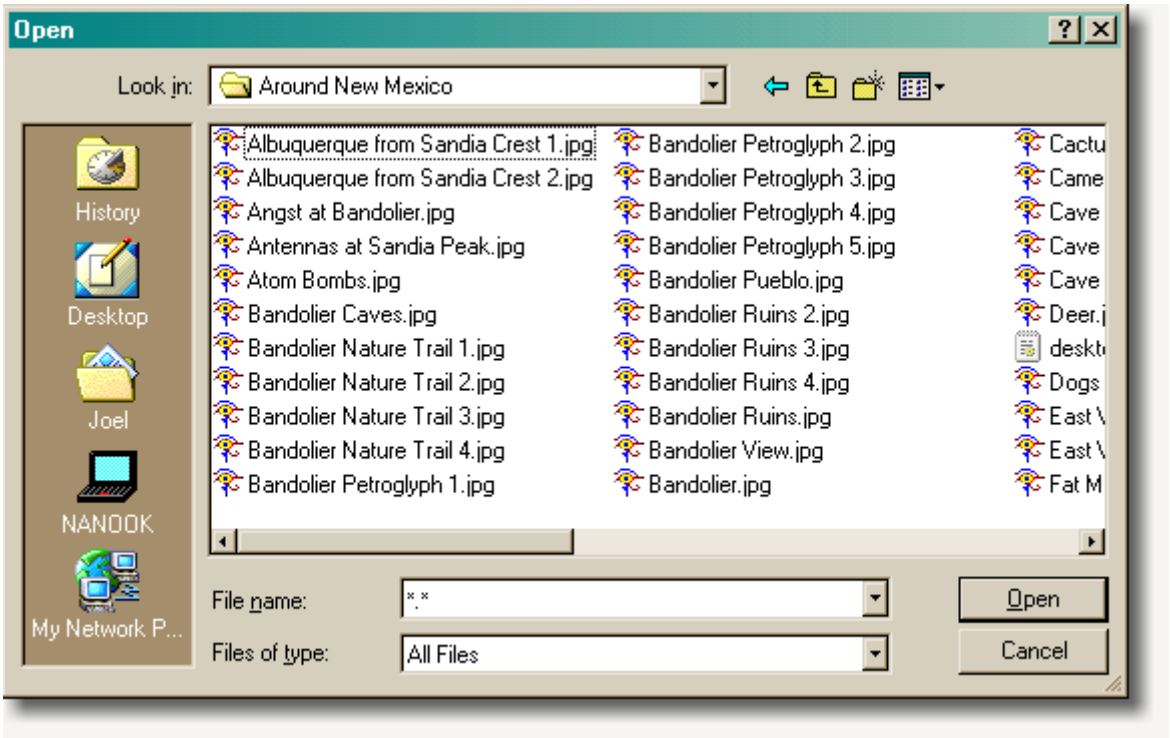
Вы лишаете пользователя возможности распоряжаться всей поверхностью экрана. Но взамен вы предлагаете интерфейс, который учитывает тот факт, что точно контролировать мышь -- сложно, и снимает с него заботу об этом. Это новшество (которое можно использовать в любой программе) интеллигентым образом уменьшает бремя управления окном. Приглядитесь к своей программе и дайте нам отдохнуть. Представьте, что мы гориллы или развитые орангутанги, и что нам трудно справиться с пронырливой мышью.

Руководство по UI дизайну для программистов
Глава8: Как создать дизайн для людей, которым есть чем заняться в этой жизни. Часть 3.

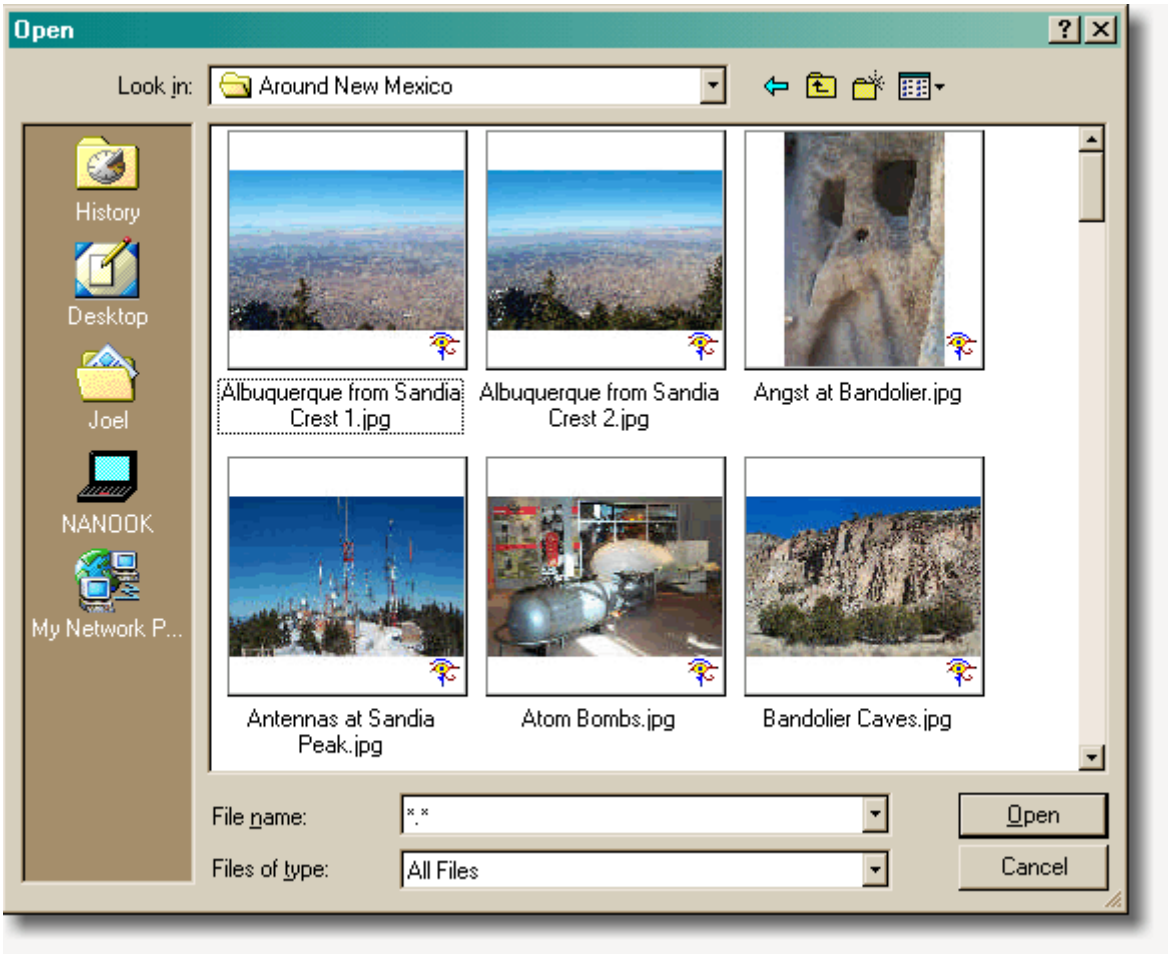
Один из самых первых принципов дизайна графических интерфейсов гласил: пользователь не должен запоминать то, что может запомнить компьютер. Классический пример - диалоговое окно *«Открыть файл»*, которое предлагает пользователю выбрать один файл из списка, а не заставляет вводить точное имя файла по памяти. Человек намного легче припоминает что-либо, когда у него есть подсказки, и всегда предпочтет выбрать из списка вместо того, чтобы выуживать необходимую информацию из глубин своей памяти.

Другим примером являются сами списки меню. Сначала был интерфейс командной строки, и вы должны были помнить все нужные вам команды. Это все равно что заставить всех желающих сделать заказ в сеульском филиале Mc'Donalds выучить корейский язык! На смену ему пришел интерфейс с полным меню, в котором перечисляются все возможные команды. И именно поэтому интерфейс командной строки не лучше графического (и неважно, что ваш друг-почитатель *UNIX* считает иначе). Интерфейс, построенный на принципе «выбрать из», подобен ресторанной карте: вы открываете ее, указываете на понравившееся блюдо и усиленно киваете головой - тот же результат, но без побочных обучающих моментов.

Посмотрите, как проходит процесс выбора файла в типичном графическом приложении:

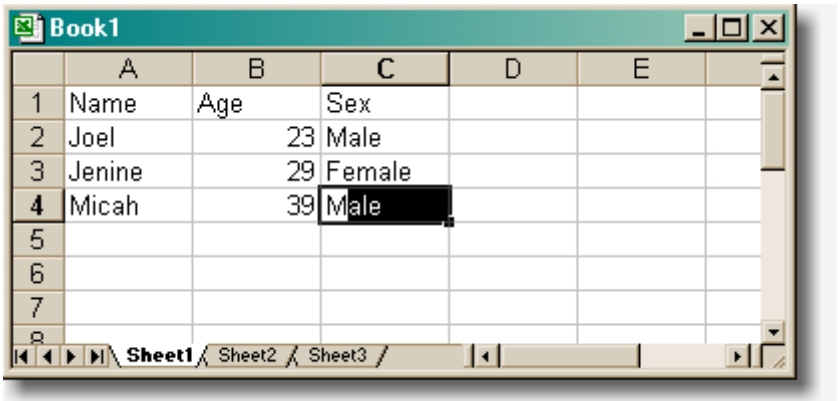


К счастью, в *Windows 98* ввели поддержку предварительного показа картинки, и вы видите те же файлы следующим образом:



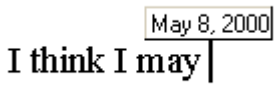
Теперь найти нужный вам файл стало значительно легче; от вас даже не требуется никаких умственных усилий на то, чтобы сопоставить слова с изображением.

Принцип минимального запоминания используется также при процессе «интеллектуального завершения ввода» (auto completion). Например, при вводе текста некоторые программы могут делать свои предположения о том, что вы собираетесь печатать:



Как только вы напечатаете «M», *Excel* предложит вариант «Male», потому что вы уже вводили этой слово в этой колонке, и может тем самым автоматически завершить процесс ввода. Если же вы хотите набрать «Mystery», а не «Male», вы просто игнорируете подсказку и вводите нужное вам слово, не тратя дополнительных усилий.

Правда, случаются и проколы, как это мог заметить любой, кто пользовался *Microsoft Word* в мае месяце:



Итак. Как создать дизайн для людей, которым есть чем заняться в этой жизни. Вкратце.

В предыдущих главах мы обсуждали следующие принципы:

- Пользователи ничего не читают (Глава 6)
- Пользователи не умеют работать с мышью (Глава 7)
- Пользователи ничего не помнят

У вас могло сложиться впечатление, что я считаю пользователей идиотами. Поверьте, это не так. Неуважение к пользователю проявляется в процессе разработки программ подобных *Microsoft Bob* (которые тут же отправляются в мусорную корзину), и выигравших в такой ситуации нет.

С другой стороны, есть и худший пример снобизма в софтверном дизайне: высокомерное предположение, что «моя софтина настолько крута, ламеры голову над ней сломают». Такое нахальство достаточно распространено в мире бесплатного прогрммного обеспечения. «Эй, *Linux* бесплатен! А если ты неспособен в нем разобраться, значит чести им пользоваться не заслуживаешь!»

Человеские способности располагаются вдоль кривой нормального распределения. Приблизительно 98% ваших клиентов достаточно развиты для того, чтобы включить телевизор. 70% из них могут пользоваться *Windows*. 15% в состоянии работать с *Linux*. 1% умеет программировать. Но лишь 0,1% владеет языком программирования уровня *C++*. И только 0,01% могут разобраться в программировании *Microsoft ATL* (и все они без исключения носят очки и бороды).

Обратный эффект этого резкого скачка состоит в том, что каждый раз когда вы даже незначительно снижаете планку (облегчаете пользование программой, скажем, на 10%), вы значительно расширяете круг потенциальных пользователей (процентов эдак на 50).

Так вот, на самом деле я не считаю людей идиотами. Но я думаю, что если вы постараетесь разработать программу, которой могут пользоваться даже идиоты, вы создадите удобную для пользователя программу, которая понравится людям и станет популярной. И вы удивитесь тому, что такие, казалось бы, незначительные улучшения привели к вам такое огромное количество новых покупателей.

Чтобы оценить, насколько удобно пользоваться программой или диалогом, который вы видите впервые, достаточно притвориться глупее, чем вы есть. Не читайте инструкций в диалоговом окне. Стройте предположения о том, что делает та или иная штуковина, и не проверяйте их. Попытайтесь пользоваться мышью одним пальцем. Делайте ошибки; валяйте дурака. Посмотрите, выполняет ли при этом программа то, что вы от нее хотите, или, по меньшей мере, дает ли она советы к действию, а не падает каждый раз, когда вы совершаете глупость. Будьте нетерпеливы. Если не удается сделать что-либо с первого раза, бросайте. И если пользовательский интерфейс не выдерживает вашего глупого, незрелого натиска, над ним стоит поработать.

Руководство по UI дизайну для программистов
Глава9: Создание продукта

Мы обсудили принципы хорошего дизайна. Но принципы могут дать возможность оценить и улучшить лишь существующий дизайн. Как же узнать, как должен выглядеть первоначальный дизайн? Многие пишут масштабные, подробные описания всех придуманных ими опций. Затем каждую разрабатывают и размещают в списке меню программы (или на веб-странице). В результате, программа (или страница) обеспечена запланированным объемом функциональности, но работа с ней отчего-то не ладится. Пользователь тупо сидит перед ней, толком не понимая, что же программа делает, не зная, как с ее помощью сделать то, что ему нужно.

В *Mictosoft* эту проблему решают на основе Планирования Деятельности. (По моим сведениям, идея принадлежит Майку Конте -- одному из разработчиков *Excel*, которому в один прекрасный момент все надоело, и он переквалифицировался в гонщики.) Суть состоит в том, чтобы предугадать те виды дейтельности, которые будет осуществлять пользователь с вашей программой, и сконцентрироваться на том, чтобы сделать выполнение деятельности простым и удобным. Рассмотрим в качестве иллюстрации следующий пример.

Вы решили сделать веб-страницу, с помощью которой можно рассылать поздравительные открытки. Пользуясь природной интуицией, вы составляете следующий список функций:

1. Написать текст
2. Добавить картинку
3. Выбрать готовую открытку из библиотеки
4. Отправить открытку:
    1. по email
    2. на печать

Нежелание или неспособность создателей приложения предоставить пользователю какой-нибудь интуитивно понятный способ обменяться данными с программой -- может закончиться созданием типичного для *Macintosh* середины 80-х интерфейса: вначале программа предлагает пустую открытку со списком меню для набора текста, картинок, для поиска и загрузки открыток из библиотеки, и для рассылки. Далее пользователь вынужден листать списки меню, пытаясь угадать все возможные комманды, и заниматься синтезом этих атомарных команд для того, чтобы создать картинку.

По методу Планирования Деятельности, вы сначала описываете те виды деятельности, которые могут заинтересовать пользователя. Достаточно побеседовать с потенциальными пользователями, и вы сможете составить такой список:

1. Поздравление с днем рождения
2. Приглашение на вечеринку
3. Поздравление с юбилеем

Теперь, вместо того, чтобы рассуждать о программе как программист («какие функции нужны для того, чтобы сделать открытку»), вы думаете как пользователь: какие действия он совершает, а именно:

1. Отправляет открытку «С днем рождения»
2. Планирует вечеринку и приглашает гостей
3. Отправляет открытку «С юбилеем»

И вдруг, рой идей проносится у вас в голове. Вместо того чтобы начать с пустой открытки, можно предложить следующие альтернативы:

Что Вы хотите сделать?

- Отправить поздравление с днем рождения
- Отправить поздравление с юбилеем
- Отправить приглашение
- Начать с пустой открытки

И вдруг окажется, что пользователю гораздо легче начать работу с программой, когда программа сама последовательно проводит его через те действия, которые нужно совершить для создания открытки. (Безусловно, существует риск того, что вы неправильно очертили круг действий, тем самым вы оттолкнете или испугаете тех, кто хотел использовать вашу программу, чтобы послать, например, открытку-поздравление с китайским новым годом, но не нашел такой опции. Поэтому отнеситесь к выбору действий самым тщательным образом, чтобы охватить большую часть людей, которых вы относите к целевой группе.)

Достаточно бросить взгляд на список основных действий, чтобы понять, что они тянут за собой дополнительные. Например, отправляя поздравление с днем рождения или юбилеем, вы можете захотеть, чтобы вам в следующем году напомнили поздравить этого человека. Поэтому уместно будеть добавить галочку «напомнить в следующем году». Приглашение же подразумевает реакцию приглашаемого, поэтому вам может прийти в голову идея, добавить опцию сбора сообщений от приглашаемых электронным путем.

Подобные возможности остаются незамеченными, когда вы разрабатываете программу, исходя из функциональности программы, а не из деятельности пользователя.

Приведенный пример тривиален; при разработке любого серьезного приложения преимущества от Планирования Дейтельности будут еще больше. Начиная с нуля разработку программы, вы уже представляете, какие виды деятельности она будет охватывать. Развить ваше представление совсем несложно, достаточно просто собрать идеи ваших коллег (посредством мозгового штурма), составить список возможных действий, и выбрать из них те, на которых вы считаете нужным сконцентрироваться. Письменное оформление ваших представлений значительно улучшит дизайн будущей программы.

Планирование Деятельности приобретает еще большее значение, когда вы работаете над второй версией программы. В таком случае полезно понаблюдать за группой пользователей, чтобы понять, в каких целях они используют программу.

На первых этапах жизни *Excel* -- до версии 4.0 -- в *Microsoft* полагали, что большая часть клиентов пользуется программой для разработки финансовых сценариев типа «что -- если», когда вы, к примеру, изменяя переменную «рост инфляции», просчитываете прибыльность предприятия.

При разработке *Excel 5.0*, когда мы впервые применили метод Планирования Деятельности, оказалось достаточным понаблюдать за тем, как пять клиентов используют программу, чтобы понять, что огромное количество людей применяют ее для ведения списков. Они не вводили никаких формул, и не совершали никаких арифметических действий! И это привело к тому, что мы придумали огромную массу функций, которые облегчали именно ведение списков: удобную сортировку, автоматическое внесение данных, автоматический фильтр для показа части списка, функции множественного пользователя, которые позволяли нескольким людям одновременно работать с одним и тем же списком при автоматическом согласовании данных.

В то время пока разрабатывался *Excel 5.0*, *Lotus* выпустил «образцовое» приложение для обработки крупноформатных таблиц, под названием *Improv*. Согласно пресс-релизам, *Improv* был программой нового поколения, которая должна была затмить все, что было придумано ранее. По странному стечению обстоятельств, *Improv* был сначала доступен на *NeXT*, что уж точно не способствовало росту продаж, но многие умные люди полагали, что *Improv* станет для *NeXT* тем, чем *VisiCalc* был для *Apple II*: приложением-приманкой, ради которого люди бросятся в магазины и закупят новое железо.

Конечно, *Improv* остался на задворках истории. Попробуйте поискать его в сети -- вы найдете его разве что на страницах архиорганизованных менеджеров складских помещений, которые -- по какой-то причине -- сделали веб-страницу с описью всего хлама, что пылится в их кладовых.

Почему? Потому что в *Improv* было практически невозможно вести списки. Его создатели полагали, что люди будут использовать приложение для создания сложных многомерных финансовых моделей. Если бы они спросили этих людей, то обнаружили бы, что ведение списков есть гораздо более распространенный вид деятельности, чем составление многоразмерных финансовых моделей, а в *Improv* ведение списков было мучительно тяжелой, а то и невыполнимой, задачей.

Итак, метод Планирования Деятельности полезен на начальной стадии развития продукта, когда нужно очертить круг действий. которые пользователи хотят выполнять, и безусловно полезен на стадии совершенствования продукта, когда нужно понять, какие действия пользователи совершают.

Еще одним примером может служить эволюция веб-страницы Deja.com, которая замышлялась как огромных поисковый портал для *Usenet*. Первоначальный интерфейс содержал всего-навсего поле для ввода текста и кнопку «искать в Usenet». Проведенное в 1999 Исследование Деятельности показало, что характерной деятельностью для большинства клиентов был сравнительный анализ продуктов или услуг, типа «какую машину выбрать». Страница была перганизована, и сегодня она представляет собой страницу по анализу отзывов о продуктах, ее изначальная поисковая функция осталась на заднем плане. Это возмутило небольшое количество клиентов, которые пользовались сайтом для поиска информации о том, совместима ли их видеокарта *Matrox* с *Redhat Linux 5.1*, но привело в восторг гораздо большую часть пользователей, которые просто хотели купить самую лучшую цифровую камеру.

Еще одна прелесть метода Планирования Деятельности состоит в том, что с его помощью вы можете составить список вещей, которыми вы заниматься не будете. При разработке какой бы то ни было программы оказывается, что список вещей, которые вы хотите сделать, в три раза больше списка вещей, которые вы успеете сделать по времени. А наилучший способ решить, что нужно сделать, и что можно не делать -- это оценить, какие функции обеспечивают решение самых важных действий пользователей.

Воображаемые пользователи.

Лучшие из лучших дизайнеров пользовательских интерфейсов единодушны в одном: перед тем как разрабатывать дизайн интерфейса, нужно придумать и описать воображаемых пользователей оного. Возможно, вы помните воображаемого пользователя Петра, которого я описал во Введении к этой книге.

Итак, Петя. Он работает бухгалтером в издательстве технической литературы, *Windows* использует уже в течение 6 лет на работе и немного дома. Компетентен, разбирается в технике. Инсталлирует программное обеспечение на своих компьютерах, почитывает журнал «PC Magazine» и как-то раз программировал несложные *Word* макросы, чтобы помочь симпатичной секретарше в бюро с рассылкой инвойсов. Дома у него кабельный модем. Петя никогда не работал с *Macintosh*. «Они слишком дорогие»,-- скажет он вам,-- «компьютер на 700 Мгц с памятью на 128 Мб можно купить за...» Да, Петр, мы поняли.

Когда вы читаете эти строки, пред вашими глазами возникает образ вашего клиента. Я мог бы придумать и иной образ:

Патриция. Профессор английского языка, автор нескольких поэтических сборников, получивших признание в определенных кругах. Использует компьютер для обработки текстов, начиная с 1980 года, хотя за все это время пользовалась лишь двумя программами: *Nota Bene* (древний академический текстовый редактор) и *Microsoft Word*. Она не собирается тратить свое время на изучение того, как работает компьютер. Имеет склонность складывать документы куда придется, как будто понятия «директория» не существует.

Совершенно очевидно, что создание программы для Петра отличается от создания программы для Патриции, которое, в свою очередь, отличается от создания программы для Макса -- шестнадцатилетнего молодца, который установил *Linux* на свом домашнем компьютере, часами болтает по «аське», и не признает софтвера от *Micro$oft*.

Когда вы придумали своих пользователей, понимание того, является ли ваш дизайн соответствующим, приходит почти автоматически. К примеру, большинство программистов склонны переоценивать способность среднестатистического пользователя догадаться о предназначении той или иной штуковины. Стоит мне написать о том, что интерфейсы командной строки неудобны в использовании, как немедленно приходит *неизбежное электронное письмо* о том, что интерфейс командной строки супер-функциональны, потому что позволяет выполнять вещи наподобие 'gunzip foo.tar.gz | tar xvf-'. Но как только вы представите себе, как Патриция будет набрать 'gunzip...', становится очевидно, что подобный интерфейс просто не может отвечать ее требованиям, никогда. Игра с воображаемым пользователем позволяет вам почувствовать своего реального пользователя, что необходимо для того, чтобы создать программу, удовлетворяющую его потребности. (И конечно, если вы разрабатываете *Linux backup* софт для продвинутых сисадминов, вам придется нарисовать образ *Серёги*, который и на пушечный выстрел не подойдет к *Windows*, который он называет исключительно «операцинной системой» в кавычках, который пользуется персонально модифицированной версией tcsh, у которого целый день работает *X11* с четырьмя рабочими столами. И с 11 xperfs впридачу.)

Подведем итог: создание хорошего софтверного продукта разбивается на шесть этапов:

1. Придумать воображаемых пользователей
2. Продумать виды деятельности пользователей
3. Узнать модель пользователя -- как он будет выполнять деятельность, основываясь на своем опыте
4. Сделать первый набросок дизайна
5. Изменять дизайн, все больше и больше делая его простым в использовании, до тех пор, пока продукт не окажется в рамках способностей воображаемых пользователей
6. Наблюдать за тем, как реальные пользователи работают с вашим продуктом. Отметить области, в которых они испытывают трудности. Эти области, скорее всего, и демонстрируют несоответствия модели программы модели пользователя.

Хороший пользовательский интерфейс -- это залог успеха на рынке. Но, кроме этого, хороший пользовательский интерфейс *делает людей немного счастливее*, потому что люди счастливы тогда, когда они решают поставленные задачи. Именно поэтому работа в сфере *UI* дизайна приносит такое удовлетворение. Где еще найдешь возможность делать миллионы людей чуточку счастливее?