



# Глава 7

## «Составные части программного интерфейса. Элементы управления»

*В этой части описаны все  
кирпичи, из которых состоит  
любая программа или сайт.*

*Это окна, элементы управления  
и стандартные сценарии  
поведения системы.*



## Тема 21. Элементы управления

1. Командные кнопки.
2. Чекбоксы и радиокнопки.
3. Списки.
4. Комбобоксы.
5. Поля ввода.
6. Крутилки.
7. Ползунки.
8. Меню.

# Командные кнопки

**Кнопкой** называется элемент управления, всё взаимодействие пользователя с которым ограничивается одним действием — нажатием.

Эта формулировка, кажущаяся бесполезной и примитивной, на самом деле очень важна, поскольку **переводит в гордое звание кнопок многие элементы управления**, которые как кнопки по большей части не воспринимаются.

Нажатие на такую кнопку запускает какое-либо явное действие, поэтому правильнее называть такие кнопки **«кнопками прямого действия»**.

Существует правило:  
**«Чем меньше кнопок, тем лучше»**.

*Правда, не всегда, так как кнопки прямого управления с точки зрения пользователя лучше, чем кнопки, раскрывающие меню, поэтому меньшее число кнопок и более простой вид лицевой панели не всегда являются лучшим решением.*

Это справедливо не только для графических интерфейсов, но и для панелей и щитков управления различных приборов, пультов управления, приборных досок и т. п.

# Командные кнопки

В то же время, этот самый простой элемент управления имеет больше всего тонкостей.

Иная ситуация в Интернете, где отсутствие операционной системы (откуда приходят элементы управления) и простота создания новых типов кнопок (это чуть ли не единственный элемент, с которым вообще удастся что-либо сделать) привели к тому, что **нестандартные кнопки не создает только ленивый**.

В Интернете кнопка должна быть оформлена как:

- ☐ *текстовая ссылка, если она перемещает пользователя на другой фрагмент текста,*
- ☐ *кнопка – если она запускает действие.*

# Командные кнопки

## Размеры и поля

Другой составляющей проблемы размера кнопок в Интернете является несоответствие видимой площади кнопки её действующей площади.

В последнее время, кнопки часто реализуют посредством окрашенных ячеек таблицы, в которых размещается текст, являющийся гипертекстовой ссылкой.

Проблема заключается в том, что пользователи воспринимают кнопку всю ячейку, хотя реально «нажимается» лишь малая её часть.

1. Как вы уже знаете, *чем больше кнопка, тем легче попасть в нее курсором*. Это правило по мере сил всеми соблюдается, во всяком случае, кнопок размером 5 на 5 пикселей уже практически не встретишь.

2. Однако помимо простоты нажатия на кнопку есть другая составляющая проблемы: *пользователю должно быть трудно нажать не на ту кнопку*. Добиться этого можно либо изменением состояния кнопки при наведении на неё курсором, либо установлением пустого промежутка между кнопками.

Первый способ приобрел существенную популярность в Интернете, второй – в обычном ПО. Он, кстати, более эффективен: одно дело, когда пользователь промахивается мимо кнопки и совсем другое – если, промахнувшись, он ещё и ошибочно нажимает на другую кнопку.

*Ни тот, ни другой способы не обеспечивают стопроцентной надежности, так что при прочих равных использовать стоит оба.*

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



# Командные кнопки

## Объем

Кнопка должна (или не должна) быть пользователем нажата. Соответственно, пользователю нужно как-то сигнализировать, что кнопка нажимаема.

Лучшим способом такой индикации является *придание кнопке псевдообъема, т.е. визуальной высоты.*

С другой стороны, этот объем плох тем, что при его использовании *возникает рассогласование между обликами кнопок прямого и непрямого действия.*

Разумеется, никто не отменял ещё и тот факт, что псевдообъем кнопок, вообще говоря, в существенной степени есть визуальный шум.

Еще, зачастую возникает необходимость максимально повышать шансы нажатия пользователем какой-либо отдельной кнопки (например, «О компании»), в этих случаях псевдообъем этой кнопки (при прочих плоских) сильно повышает вероятность нажатия.

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

# Командные кнопки

## Состояния

Вывод:

- ☐ не должно быть разных состояний, выглядящих одинаково,
- ☐ заблокированные состояния действительно важно делать заблокированными.

Например, в Интернете очень часто встречаются кнопки, нажатие на которые открывают ту же самую страницу, т.е. нажатие которых возможно, но бесполезно. Такие кнопки должны не только выглядеть заблокированными (менее яркими и значительными, нежели обычные), но и не нести гипертекстовых ссылок.

Кнопка должна как-то *показывать пользователям свои возможные и текущие состояния.*

Количество состояний довольно велико, при этом наборы возможных состояний в ПО и в Интернете значительно различаются.

Например, кнопка в Windows может иметь пять состояний:

- ☐ нейтральное,
- ☐ нажатое,
- ☐ нейтральное с установленным фокусом ввода,
- ☐ состояние кнопки по умолчанию?
- ☐ заблокированное состояние.

В Интернете обычно используют меньший набор состояний: нейтральное,

- ☐ готовое к нажатию (onMouseOver)?
- ☐ активное (в случаях, когда набор кнопок используется для индикации навигации).

Нажатое и заблокированное состояние используются очень редко, а «нейтральное с установленным фокусом ввода» старается, как может, создать браузер.

Вообще говоря, обычно, чем больше набор состояний, тем лучше.

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

# Командные кнопки

## Текст и пиктограммы

Достоинством глагольных кнопок то, что по ним понятно, какое действие произойдет после нажатия.

Все руководства по разработке интерфейса с изумительным упорством *требуют снабжать командные кнопки названиями, выраженными в виде глаголов в форме инфинитива.*

Разработчики же интерфейса с не менее изумительным упорством не следуют этому правилу. Аргументов у них два:

- ☐ во-первых, все так делают, значит, это есть стандарт и ему нужно следовать,
- ☐ во-вторых, нет времени придумывать название.

Оба аргументы сильны. Действительно, стандарт. Действительно, нет времени.

Но есть два контраргумента:

- ☐ во-первых, это не столько стандарт, сколько стандартная ошибка,
- ☐ во-вторых, думать можно и по дороге домой.

Если второй контраргумент особых объяснений не требует, то сущность первого полезно объяснить.

Кнопка, запускающая действие, называется командной т.к. с её помощью пользователи отдадут системе команды.

*Команда же в русском языке формируется посредством глагола в повелительном наклонении.*



# Командные кнопки

## Текст и пиктограммы

Как правило, разработчики создают диалоговое окно, внизу которого располагают три кнопки:

- ☐ «Ввод»,
- ☐ «Применить»
- ☐ «Отмена»

(прямо-таки триединство ошибки).

Таким образом, *следует всемерно избегать создания кнопок с ничего не говорящим текстом*, поскольку такой текст не сообщает пользователям, что именно произойдет после нажатия кнопки.

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

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

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

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

Если бы вместо кнопки *Применить* была бы кнопка *Предварительный просмотр*, все бы работало великолепно.



# Командные кнопки

## Текст и пиктограммы

*Пиктограмма (icons) — это маленькие картинки, служащие для обозначения кнопок и других объектов.*

Помимо текста, на кнопках можно выводить пиктограммы.

Однако у пиктограмм есть существенный недостаток — это затруднительное смысловое распознавание.

В последних версиях операционных системах Macintosh и Windows при наведении курсора на пиктограмму появляется окно с текстом ее описания. Появляется естественный вопрос. Почему не использовать текст без пиктограммы?

Формально, на таких кнопках пиктограммы не очень хороши из-за того, что они обычно должны передавать пользователям идею *действия* (т.е. глагол), а *действие плохо передается пиктограммами.*

Конечно, даже и нераспознанная пиктограмма хороша тем, что она визуально отделяет кнопку от кнопки и для опытных пользователей обеспечивает ускорение при поиске нужной кнопки (пользователь может помнить, что ему нужна кнопка с синим пятном на пиктограмме). *Пиктограмма будет полезной только тогда, когда она грамотно исполнена и легко читаема любым пользователем.*

Так что, *пиктограммы хороши для тех кнопок, для которых пиктограммы нарисовать легко.*

Для тех кнопок, которые нужны особенно часто, качество пиктограммы может особого значения не иметь, важно только различия пиктограмм между собой.

# Командные кнопки

## *Кнопки доступа к меню*

Также к группе командных кнопок относится кнопка доступа к меню.

Формально, это попытка скрестить ужа (раскрывающийся список) с ежом (кнопкой), но попытка удачная.

Существует много ситуаций, когда раскрывающийся список не помещается в отведенное для него место, поскольку текст в списке слишком велик. Для этого необходимо вставить кнопку, нажатие на которую будет вызывать меню.

Однако:

Во-первых, недостаток кнопки будет проявляться в том, что, поскольку по условиям задачи, текст не будет виден, значение кнопки будет менее понятным, чем контекстное меню безо всякой кнопки. Формально, для совсем уж неопытных пользователей, кнопка работать будет, но, как только пользователи подрастут, контекстное меню окажется эффективней. Как никак, в кнопку надо попасть курсором, а в меню попадать не надо, достаточно просто нажать правую кнопку мыши, огорчает также и то, что кнопки загромождают экран.

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

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

# Командные кнопки

## Кнопки доступа к меню

Существуют, впрочем, определенные ситуации, когда такие кнопки очень хороши.

Только нужно сделать так, *чтобы кнопка была одновременно и командной кнопкой, и показывала меню.*

Для этого :

- ❑ Во-первых, нужно разделить кнопку на две области, одна из которых запускает действие, а другая открывает меню.
- ❑ Во-вторых, нужно организовать такой контекст, при котором результат нажатия на кнопку всегда будет понятным.

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

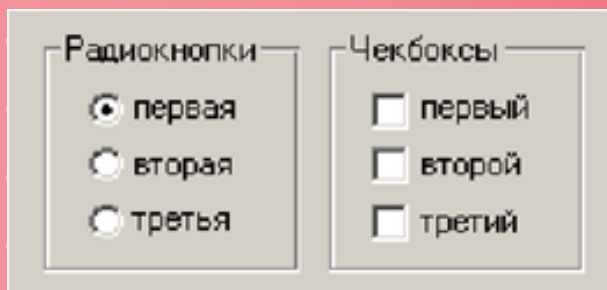
Иногда бывают ситуации, когда действий может выполняться несколько, но чаще всего нужно только одно. В этом случае пользователи очень быстро обучаются этому действию, имея довольно простой доступ к остальным. В таком исполнении кнопки доступа к меню работают замечательно.

Дополнительно:

- ❑ Во-первых, *на области, вызывающей меню, обязательно должно находиться изображение направленной вниз стрелки.*
- ❑ Во-вторых, *эта область должна находиться справа на кнопке, чтобы изображение стрелки не мешало воспринимать текст или пиктограмму на кнопке.*



# Чекбоксы и радиокнопки



The image shows a UI mockup with two groups of controls. The first group, titled 'Радиокнопки' (Radio buttons), contains three radio buttons labeled 'первая' (first), 'вторая' (second), and 'третья' (third). The second group, titled 'Чекбоксы' (Checkboxes), contains three checkboxes labeled 'первый' (first), 'второй' (second), and 'третий' (third).

## Общее

☐ Они являются **кнопками отложенного действия**, т.е. их нажатие не должно инициировать какое-либо немедленное действие.

☐ С их помощью пользователи вводят параметры, которые скажутся после, когда действие будет запущено иными элементами управления.

*Нарушать это правило опасно*, поскольку это серьезно нарушит сложившуюся ментальную модель пользователей.

## Различие

☐ Группа чекбоксов даёт возможность пользователям выбрать любую комбинацию параметров.

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

☐ Радиокнопки позволяют выбрать только один параметр.

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

Это сближает эти элементы со списками множественного и единственного выбора соответственно.

# Чекбоксы и радиокнопки

Например, если нужно спросить пользователя, в какой кодировке посылать ему письма, не получится заменить две радиокнопки Windows 1251 и KOI-8 единым чекбоксом KOI-8. Пользователь не обязан понимать, в какой кодировке система будет посылать ему письма по умолчанию. К счастью, такие ситуации редки.

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

Всякий раз, когда пользователю нужно предоставить выбор между несколькими параметрами, можно использовать либо чекбоксы, либо радиокнопки (или списки, но о них позже).

1. Если параметров больше двух, выбор прост:

- ☐ если параметры можно комбинировать, нужно использовать чекбоксы (например, текст может быть одновременно *и* жирным *и* курсивным);
- ☐ если же параметры комбинировать нельзя, нужно использовать радиокнопки (например, текст может быть выровнен *или* по левому, *или* по правому краю).

2. Если же параметров всего два и при этом параметры невозможно комбинировать (т.е. либо ДА, либо НЕТ), решение более сложно. Дело в том, что группу из двух радиокнопок часто можно заменить одним чекбоксом. К сожалению, этот метод работает не всегда.

# Чекбоксы и радиокнопки

## Внешний вид.

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

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

## Текст подписей.

❑ Каждая подпись должна однозначно показывать эффект от выбора соответствующего элемента.

❑ Поскольку радиокнопки и чекбоксы не вызывают немедленного действия, *формулировать подписи к ним лучше всего в форме существительных, хотя возможно использование глаголов* (если изменяется не свойство данных, а запускается какое-либо действие).

❑ *Подписи к стоящим параллельно кнопкам лучше стараться делать примерно одинаковой длины.*

❑ *Все подписи обязаны быть позитивными* (т.е. не содержать отрицания).

❑ *Повторять одни и те же слова, меняя только окончания подписей* (например, «Показывать пробелы» и «Показывать табуляции»), *в нескольких кнопках нельзя*, в таких случаях лучше перенести повторяющееся слово в рамку группировки.

❑ *Если подпись не помещается в одну строку, выравнивайте индикатор кнопки* (кружок или квадрат) *по первой строке подписи.*



# Чекбоксы и радиокнопки

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

Взаимодействие.

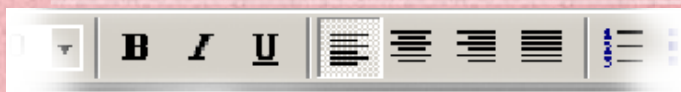
*Это может показаться невероятным, но до сих пор в интернете 99% чекбоксов и радиокнопок реализованы неправильно.*

Дело в том, что создатели языка HTML, ничего не понимавшие в проектировании интерфейсов, были поначалу искренно уверены в том, что в этих элементах управления нажимается только визуальный индикатор переключения, т.е. кружок или квадратик. На самом деле это совершенно не так! *Нажимабельной должна быть ещё и подпись, просто потому, что закон Фитса* (см. «Быстрый или точный») однозначно требует больших кнопок. Но в Интернете всего этого нет, поскольку в HTML конструкция чекбоксов и радиокнопок просто не позволяет делать нажимаемыми подписи. Сейчас это стало технически возможным (через тег Label), но по инерции и вполне понятной лени никто чекбоксы нормальными не делает.

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

# Чекбоксы и радиокнопки

Пример чекбоксов и радиокнопок на панели инструментов:



- ☐ Слева расположены чекбоксы (шрифт может быть и жирным и курсивным),
- ☐ справа радиокнопки (абзац может быть выровнен либо по левому, либо по правому краю).

*Вариант для панелей инструментов*

Как чекбоксы, так и радиокнопки, бывают двух видов:

- ☐ Стандартные (описанные выше),
- ☐ Предназначенные для размещения на панелях инструментов.

Визуально чекбоксы и радиокнопки не различаются.

У них есть определенный недостаток: они не различаются внешне (насколько известно, ни в одной ОС метода визуального различения не выработано).

Тем не менее, *на панелях инструментов полезно располагать группы радиокнопок отдельно от групп чекбоксов* (чтобы они не смешивались в сознании пользователей).

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

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

# Списки

Все часто используемые списки функционально являются вариантами чекбоксов и радиокнопок.

Списки бывают:

- ☐ пролистываемыми (могут обеспечивать как единственный (аналогично группе радиокнопок), так и множественный выбор (чекбокс);
- ☐ раскрывающимися (работают исключительно как радиокнопки).

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



# Списки

## Общие свойства всех списков

❑ Ширина списка как минимум должна быть достаточна для того, чтобы пользователь мог определить различия между элементами.

*В идеале, конечно, ширина всех элементов должна быть меньше ширины списка*, но иногда это невозможно. В таких случаях *не стоит добавлять к списку горизонтальную полосу прокрутки, лучше урезать текст элементов*. Для этого нужно определить самые важные фрагменты текста (например, для URL это начало и конец строки), после чего все остальное заменить отточием (...).

❑ Поскольку нужно максимально ускорить работу пользователей, необходимо сортировать элементы.

*Идеальным вариантом является сортировка по типу элементов.*


*Если же элементы однотипны, их необходимо сортировать по алфавиту, причем списки с большим количеством элементов полезно снабжать дополнительными элементами управления, влияющими на сортировку или способ фильтрации элементов.*

❑ Если можно определить наиболее популярные значения, их можно сразу расположить в начале списка, но при этом придется вставлять в список разделитель, а в систему – обработчик этого разделителя.



# Списки


## Пиктограммы



Уже довольно давно в ПО нет технических проблем с выводом в списках пиктограмм отдельных элементов.

Однако практически никто этого не делает.

Это плохо, ведь пиктограммы обеспечивают существенное повышение субъективной привлекательности интерфейса и сканируются быстрее простого текста.



# Списки

## Раскрывающиеся списки

Самым простым вариантом списка является *раскрывающийся список*



Смещение:  на:

Пример визуальной сборки команды из составляющих. Данный метод значительно проще для понимания, нежели, например, ввод положительного значения для смещения вверх и отрицательного значения для смещения вниз без поддержки раскрывающимся списком.

Малая высота раскрывающегося списка позволяет с большой легкостью визуально отображать команды, собираемые из составляющих.

Раскрывающийся список, как правило, вызывает две проблемы:

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


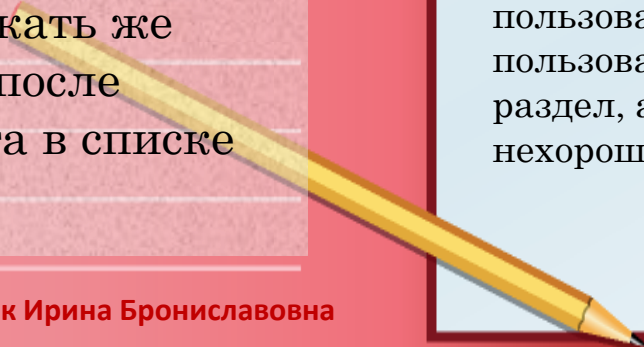




# Списки

## *Раскрывающиеся списки*


Таким образом,  
навигационные  
раскрывающиеся списки  
нужно снабжать кнопкой,  
которая и будет запускать  
действие, запускать же  
действие сразу после  
выбора элемента в списке  
нельзя.



## 2. другая – в Интернете.

Раскрывающийся список часто используется как навигационное меню. Это изначально неправильно, поскольку содержимое такого меню не видно сразу и уж тем более им трудно индцировать пользователям, в каком разделе сайта они находятся. Это, впрочем, не главное.

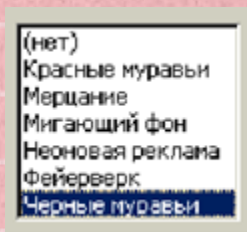
Большая проблема заключается в том, что список снабжают скриптом, который запускается сразу по выбору значения. Такой метод имеет два недостатка:

- ❑ Во-первых, список исторически не является элементом управления прямого действия (как и чекбокс, например), что приводит к потере пользователями чувства контроля над системой.
  - ❑ Во-вторых, раскрывающиеся списки довольно сложны, так что пользователи часто совершают моторные ошибки при выборе нужного элемента. Поскольку эта ошибка не может быть обнаружена системой и не всегда обнаруживается пользователями, часты ситуации, когда пользователь (как ему кажется) выбирает один раздел, а перемещается в другой, что совсем нехорошо.
- 

# Списки

## Пролистываемые списки

Другим, более сложным вариантом списка является пролистываемый список.



В кругу интеллектуалов слово "моветон" иногда в шутку используют в отношении примитивных суждений и расхожих стереотипов, которые явно диссонируют с общим уровнем разговора, чрезмерно упрощая предмет беседы и искажая его характеристики.

Пролистываемые списки могут позволять пользователям совершать как единственный, так и множественный выбор.

Одно требование применимо к обоим типам списков, остальные применимы только к одному типу.

### Размер.

*По вертикали в список должно помещаться как минимум четыре строки, а лучше восемь.*

Напротив, *список*, по высоте больший, нежели высота входящих в него элементов, и соответственно, *содержащий пустое место в конце, смотрится неряшливо.*

Требование *выводить полосы прокрутки в больших списках кажется моветоном*, но забывать о нем не следует.

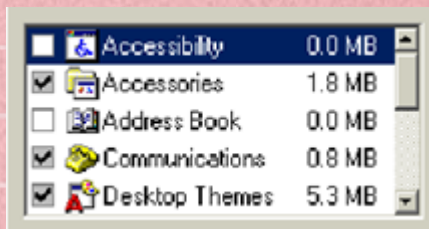
### Списки единственного выбора.

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

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

# Списки

## Списки множественного выбора



С точки зрения дизайна интерфейсов, *списки множественного выбора* интересны, прежде всего, тем, что *их фактически нет в Интернете.*

Технически создать список множественного выбора не проблематично, для этого в HTML есть даже специальный тег. Проблема в том, что такой список в браузере будет выглядеть как список единственного выбора, более того, *чтобы выбрать несколько элементов пользователю придется удерживать клавишу Ctrl.*

Это значит, что воспользоваться таким списком сможет только малая часть аудитории (и даже наличие подсказки у списка положения не исправит).

*Из-за такой убогой реализации списков браузерами, использовать их, как правило, оказывается невозможно.*

Приходится использовать чекбоксы.

Гораздо лучше обстоят дела в ПО.

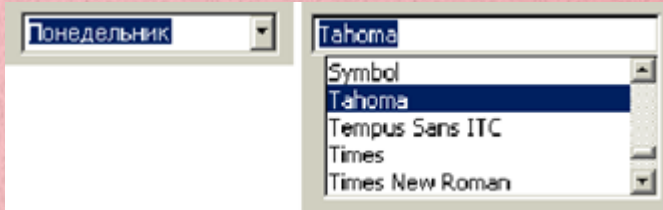
*Возможность безболезненно выводить в списке чекбоксы позволяет пользователям без труда пользоваться списками, а разработчикам – без труда эти списки создавать.*



# Комбобоксы

## Комбобоксами

(combo box), называются гибриды списка с полем ввода, когда пользователь может выбрать существующий элемент, либо ввести свой.



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

Комбобоксы бывают двух видов:

- ☐ раскрывающиеся,
- ☐ расширенные.

Оба типа имеют проблемы.

## Проблемы раскрывающегося комбобокса.

Во-первых, такие комбобоксы выглядят в точности как раскрывающиеся списки, визуально отличаясь от них только наличием индикатора фокуса ввода (да и то, только тогда, когда элемент выделен).

Это значит, что полноценно пользоваться ими могут только сравнительно продвинутые пользователи. В этом нет особой проблемы, поскольку комбобоксом все равно можно пользоваться, как обычным списком.

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

## Проблемы расширенных комбобоксов.

Их с трудом, но можно реализовать в Интернете (через JavaScript). Они имеют уникальный вид, отличающий их от остальных элементов управления. Зато их сравнительно трудно (хотя и гораздо легче, чем в Интернете) реализовать в ПО, поскольку в Windows нет такого элемента, так что собирать его приходится из двух. При этом расширенный комбобокс потребляет много места на экране.

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

# Поля ввода

Вместе с командными кнопками, чекбоксами и радиокнопками, поля ввода являются основой любого интерфейса.

Рассмотрим предъявляемые к ним требования.

## Размеры.

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

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

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

***Ширина поля ввода не должна быть больше максимальной длины строки.***

# Поля ввода

Всякий раз, когда ширина поля ввода больше максимального объема вводимого в него текста, при этом объем вводимого текста ограничен, пользователи неприятно изумляются, обнаружив, что они не могут ввести текст, хотя место под него на экране имеется.

**Соответственно, вообще нельзя делать поле ввода шире максимального объема вводимого в них текста.**

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

## Код активации

Введите оставшуюся часть кода активации (без серийного номера).  
На Интернет-карте сотрите защитную полосу, чтобы прочитать код

7710812020 -  -  -  -

Пример полей ввода, больших объема вводимых в них информации. Такие поля *выглядят неряшливо и вдобавок обманывают пользователей, показывая, что пользователь ввел не всю информацию. причем, после заполнения поля приходится самому перемещать фокус ввода, хотя с этим справилась бы и система.*



# Поля ввода

## Подписи.

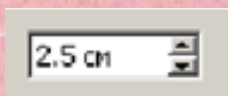
Вопрос «*где надо размещать подписи к полям ввода?*» является одним из самых популярных среди программистов.

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

Подписи к полям ввода имеют определенное отличие от других подписей. *В полях ввода подписи можно размещать не рядом с элементом, а внутри него, что позволяет экономить пространство экрана.* Подпись при этом выводится в самом поле ввода, точно так же, как и текст, который в него нужно вводить. Необходимо только отслеживать фокус ввода, чтобы при установке фокуса в поле убирать подпись. Это решение, будучи нестандартным, плохо работает в ПО, но неплохо работает в Интернете. Если очень жалко экранное пространство, этим методом стоит пользоваться.

# Крутилки

**Крутилка** (spinner, little arrow) есть поле ввода, однако не такое универсальное, как обычное т.к. не позволяет вводить текстовые данные.



Основные возможности крутилки.

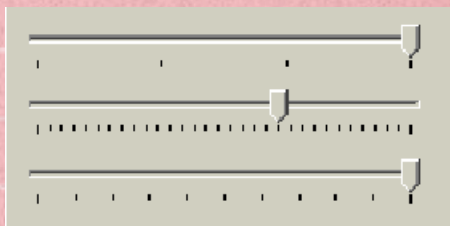
- ❑ Во-первых, чтобы ввести значение в крутилку, пользователю *не обязательно бросать мышь и переносить руку на клавиатуру* (в отличие от обычного поля ввода). Поскольку перенос руки с места на место занимает сравнительно большое время (в среднем почти половину секунды), к тому же ещё и сбивает фокус внимания, отсутствие нужды в клавиатуре оказывается большим благом. Во всяком случае, ввод значения в крутилку с клавиатуры достаточно редок, т.е. пользователи воспринимают крутилки целиком и полностью положительно.
- ❑ Во-вторых, при вводе значения мышью система может позволить пользователям вводить только *корректные данные, причем, что особенно ценно, в корректном формате*. Это резко уменьшает вероятность человеческой ошибки.

Таким образом, использование крутилок для ввода любых численных значений более чем оправдано.

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

# Ползунки

Как и ранее описанные элементы управления, *ползунки позволяют пользователям выбирать значение из списка, не позволяя вводить произвольное значение.*



Возникает резонный вопрос: *зачем нужен ещё один элемент управления, если аналогичных элементов уже полно?*

Ответ прост: *ползунки незаменимы, если пользователям надо дать возможность выбрать значение, стоящее в хорошо ранжирующемся ряду, если:*

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

Ползунки имеют интересный аспект.

Их можно также использовать для выбора текстовых параметров, но только в случаях, когда эти параметры можно понятным образом отранжировать.

Случаев таких немало, например, «завтрак», «обед» и «ужин», при отсутствии внешней связи ранжированию поддаются вполне.



# Меню

**Меню** – это метод взаимодействия пользователя с системой, при котором пользователь выбирает из предложенных вариантов, а не предоставляет системе свою команду.

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

В настоящее время систем, которые не использовали бы меню в том или ином виде, практически не осталось. Объясняется это просто.

Меню:

- ❑ позволяет *снизить нагрузку на мозги пользователей*, поскольку для выбора команды не надо вспоминать, какая именно команда нужна и как именно её нужно использовать – вся (или почти вся) нужная информация уже содержится на экране;
- ❑ ограничивает диапазон действий пользователей, появляется возможность в значительной мере *изъять* из этого диапазона *ошибочные действия*;
- ❑ показывает пользователям объем действий, которые они могут совершить благодаря системе, и тем самым *обучают пользователей* (в одном из исследований было обнаружено даже, что меню является самым эффективным средством обучения).

Таким образом, в большинстве систем меню является объективным благом (*они неэффективны, в основном, в системах с внешней средой или течением времени*).

# Меню

## Типы меню

Существуют несколько различных классификаций меню, но основной интерес представляют только две из них.

Первая классификация делит меню на два типа:

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

В некоторых ситуациях эти два типа меню могут сливаться в один: например, меню, состоящее из кнопок доступа к меню, могут работать и как статические (пользователи нажимают на кнопки) и как динамические (пользователи вызывают меню).

Вторая классификация также делит меню на два типа:

- ❑ **разворачивающиеся в пространстве** (например, обычное выпадающее меню). Всякий раз, когда пользователь выбирает элемент нижнего уровня, верхние элементы остаются видимыми.
- ❑ **разворачивающееся во времени**. При использовании таких меню элементы верхнего уровня (или, понимая шире, уже пройденные элементы) по тем или иным причинам исчезают с экрана.

# Меню

При разработке интерфейса необходимо использовать принцип **адаптивного меню**.

Меню делается адаптивными исходя из того предположения, что если оставлять часто используемый элемент на виду, без необходимости поиска его в меню, то это может ускорить работу пользователя.

Рассмотрим недостатки каждого типа меню. Статические меню, как правило, обеспечивают высокую скорость работы, лучше обучают пользователей, но зато *занимают место на экране*.

Меню, разворачивающиеся в пространстве, обеспечивают большую поддержку контекста действий пользователей, но эта поддержка обходится в *потерю экранного пространства*.

Меню, разворачиваемое во времени, более бережно использует пространство, но зато *хуже поддерживает контекст*.

Особо необходимо отметить меню типа «*мастер*». Являясь и динамическим и разворачивающимся во времени меню оно не оказывается более быстрым, чем, например, раскрывающееся меню. Но объем и специфика входящих в него элементов управления не позволяют, как правило, сделать из него какое-либо другое меню, например, раскрывающееся.

Вывод: *очень полезно научиться анализировать влияние и взаимопроникновение разных типов меню, а также осознавать их место в интерфейсе*.



# Меню

## Устройство отдельных элементов

К наименованию элементов меню нужно подходить весьма тщательно, тщательней, нежели ко всему остальному.

*Выбранное название должно быть понятно целевой аудитории.*

*Не стоит уместать в диалоговое окно какую-либо функцию, если её существование в этом окне невозможно предсказать, глядя на соответствующий элемент меню.*

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

*Элементы, запускающие действия, должны быть глаголами в форме инфинитива (как командные кнопки). Впрочем, часто глагол приходится выкидывать вообще, чтобы переместить значимое слово ближе в начало текста элемента. Нужно это, чтобы повысить скорость распознавания. Повысить её можно всего одним способом: **главное** (т.е. наиболее значимое) **слово в элементе должно стоять в элементе первым**. Обратите внимание, что короткий текст элемента, без сомнения, быстро читаясь, совершенно необязательно быстро распознается. Поэтому **не стоит безудержно сокращать текст элемента: выкидывать нужно все лишнее, но не более**.*

# Меню

## Пиктограммы в меню

*Не снабжайте пиктограммами все элементы меню, снабжайте только самые важные.*

*Пиктограммы в меню*, если они повторяют пиктограммы в панели инструментов, *обладают замечательной способностью обучать пользователей возможностям панели.*

Помимо этого они *здорово ускоряют поиск известного элемента и точность его выбора*, равно как и общую разборчивость меню.

Таким образом, пиктограммы в меню объективно хороши (только стоят дорого, к сожалению). Это очевидный факт.

Теперь менее очевидный: *пиктограммы лучше работают, когда ими снабжены не все элементы.*

Когда все элементы имеют пиктограммы, разборчивость каждого отдельного элемента падает: в конце концов, *пиктограммы всех ненужных в данное время элементов являются визуальным шумом.*

# Меню

## Переключаемые элементы

*Всегда формулируйте  
текст в интерфейсе без  
использования  
отрицаний.*

Особого внимания заслуживают случаи, когда меню переключает какие-либо взаимоисключающие параметры, например, показывать или не показывать палитру.

Тут есть несколько возможных способов.

- ☐ Можно *поместить перед переключателем галочку, показывая, что он включен* (если же элемент снабжен *пиктограммой, можно её утапливать*). Этот метод считается лучшим.
- ☐ Можно не помещать галочку, зато *инвертировать текст элемента*: например, элемент *Показывать сетку* превращается в *Не показывать сетку*. Это плохо по многим причинам.
  - Во-первых, в интерфейсе желательно не употреблять ничего негативного: в меньшей степени потому, что негативность слегка снижает субъективное удовлетворение; в большей степени потому, что она снижает скорость распознавания текста (главное слово не первое, нужно совершить работу, чтобы из отрицания вычислить утверждение).
  - Во-вторых, если изъять «не» и переформулировать одно из состояний элемента, пользователям будет труднее осознать, что два разных элемента на самом деле есть один элемент.

Таким образом, *галочка предпочтительнее*.



# Меню

## Предсказуемость действия

Пользователей нужно снабжать чувством контроля над системой.

Применительно к меню это значит, что *по виду элемента пользователи должны догадываться, что произойдет после выбора.*

Сделать это невероятно трудно, поскольку на экране нет места под такие подсказки. Можно сделать только одно, но сделать это нужно обязательно: нужно показать пользователям, какой элемент запускает действие или меняет параметр, а какой открывает окно с продолжением диалога.

Почти во всех ОС *стандартным индикатором продолжения диалога является многоточие после текста элемента, так что пользоваться этим признаком стоит везде, включая Интернет.*

Также *необходимо показывать, какой элемент срабатывает сразу, а какой открывает элементы меню нижнего уровня* (в любой ОС это делается автоматически, в Интернете нужно не забывать делать это вручную). Это же правило касается и гипертекстовых ссылок вообще (они тоже меню).

Пользователи испытывают значительно большее чувство контроля, когда имеют возможность предсказать, куда их ссылка приведет (при этом снижается количество ошибочных переходов).

Таким образом, *нестандартные ссылки* (т.е. ссылки на другой сайт, на почтовый адрес, на файл, на узел FTP, на долго загружающуюся страницу и т.д.) *полезно снабжать характерными для них признаками*, например, ссылку на почтовый адрес пиктограммой письма.

# Меню

## Группировка элементов

Второй *составляющей*  
*качества меню является*  
*группировка его*  
*элементов.*

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

Чтобы уметь эффективно группировать элементы в меню, нужно знать ответы на три вопроса:

1. **Зачем элементы в меню нужно группировать?**
2. **Как группировать элементы ?**
3. **Как разделять группы между собой?**

# Меню

Зачем элементы в меню  
нужно группировать?

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



# Меню

## Как группировать элементы?

- а) Каждый знает, или, во всяком случае, догадывается, что *элементы в меню нужно группировать максимально логично*. Пospорить с этим утверждением нельзя, но от этого его проблематичность не уменьшается.
- б) *Взаимоисключающие элементы желательно помещать в отдельный уровень иерархии*.

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

Для этого используется очень простой и надежный метод, называемый *карточной сортировкой*.

# Меню

Как разделять группы между собой?

Существует два основных способа разделять группы:

- ☐ между группами можно помещать пустой элемент (разделитель),
- ☐ размещать отдельные группы в разных уровнях иерархии.

Второй способ создает более четкое разделение: в меню Файл, например все элементы более близки друг другу (несмотря на разделители), чем элементы других меню.

В то же время выбор конкретного способа диктуется результатами карточной сортировки, так что интерес представляет только вопрос «как должны выглядеть и действовать разделители».

Для *разграничения групп традиционно используют полосы*. Это надежное, простое решение, другой разговор, что *с дизайнерской точки зрения полосы плохи, поскольку представляют собой визуальный шум*.

Гораздо правильнее, но и труднее, *использовать только визуальные паузы* между группами, как это сделано, например, в MacOS X.

# Меню

## Глубина меню


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

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






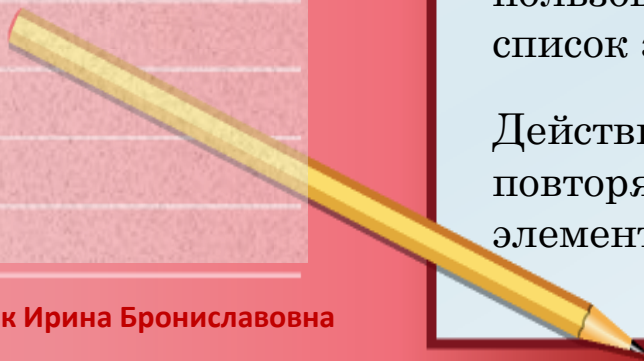
# Меню



При перемещении по меню пользователь действует по определенному алгоритму:

1. Выбирая элемент первого уровня, он выбирает элемент, «нужность» которого кажется ему максимальной.
2. После выбора он видит список элементов второго уровня, при этом он оценивает вероятность соответствия всех элементов второго уровня его задаче и одновременно выбирает наиболее вероятный элемент. При этом в уме он держит контекст, т.е. название элемента первого уровня.
3. Если ни один из элементов не кажется пользователю достаточно вероятным, пользователь возвращается на первый уровень.
4. Если какой-то элемент удовлетворяет пользователя, он выбирает его и получает список элементов третьего уровня.


Действия из второго и третьего шагов повторяются применительно к новым элементам меню.





# Меню

## Вывод




В целом, ширина и глубина меню являются, пожалуй, наименее значимыми факторами.

Гораздо важнее хорошая группировка, при этом как группировку, так и структуру дерева меню, всё равно лучше определять карточной сортировкой.

Применительно же к раскрывающимся меню действует ещё один ограничитель глубины. Раскрывающиеся меню довольно тяжелы в использовании, поскольку требуют от пользователей достаточно тонкой моторики.

Поэтому *главное меню с уровнем вложенности элементов большим трех просто невозможно.*



# Меню

## Контекстные меню

*Не делайте контекстные  
меню единственным  
способом вызова какой-  
либо функции*

Преимущество контекстных  
(всплывающих) меню:

- ☐ *они полностью встраиваются в  
контекст действий  
пользователей: не нужно  
переводить взгляд и курсор в  
другую область экрана,*
- ☐ *практически не нужно  
прерывать текущее действие  
для выбора команды,*
- ☐ *не занимают места на экране.*

Недостаток: из-за того, что они не  
находятся всё время на экране, они  
практически неспособны чему-либо  
научить пользователя.



# Меню

## Контекстные меню

Основная причина появления контекстных меню: стремление максимально повысить скорость работы пользователей, поэтому на их **размер и степень иерархичности** накладываются определенные ограничения.

Если меню будет длинным, пользователям придется сравнительно долго возвращать курсор на прежнее место, так что привлекательность нижних элементов окажется под вопросом.

Поэтому *лучше сокращать размер контекстных меню до разумного минимума (порядка семи элементов).*

К тому же не надо забывать, что главное меню не всегда перекрывает выделенный (т.е. актуальный объект), а контекстное меню – почти всегда (как-никак оно вызывается на самом объекте).

В большинстве же случаев *перекрытие актуального объекта нежелательно* (сбивается контекст).

Другая особенность контекстных меню – *иерархия*.


В обычном меню иерархия имеет хотя бы одно достоинство: при обучении она позволяет упорядочивать элементы меню и тем самым делать его понятнее. В контекстных же меню обучающая функция не играет никакой роли, поскольку такими меню пользуются только опытные пользователи.

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



# Меню

## Контекстные меню



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

