Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Институт повышения квалификации
и переподготовки кадров

Кафедра «Информатика»

А. И. Рябченко, В. О. Лукьяненко

ИНФОРМАЦИОННАЯ АРХИТЕКТУРА И ЮЗАБИЛИТИ

КУРС ЛЕКЦИЙ
по одноименной дисциплине для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заочной формы обучения

Гомель 2012

УДК 004.738.12(075.8)
ББК 32.81я73
Р98

Рекомендовано кафедрой «Информатика» ГГТУ им. П. О. Сухого
(*протокол № 12 от 06.06.2012 г.*)

Рецензент: проф., д-р техн. наук, проф. каф. «Информационные технологии»
ГГТУ им. П. О. Сухого *И. А. Мурашко*

Рябченко, А. И.

Р98 Информационная архитектура и юзабилити : курс лекций по одноим. дисциплине для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заоч. формы обучения / А. И. Рябченко, В. О. Лукьяненко. – Гомель : ГГТУ им. П. О. Сухого, 2012. – 109 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://alis.gstu.by/StartEK/. – Загл. с титул. экрана.

Описаны этапы становления человеко-компьютерного взаимодействия как отдельной дисциплины. Рассматриваются общие принципы взаимного расположения объектов на веб-странице, их размеры и выбор цветовой палитры, описаны средства привлечения внимания пользователей. Приведена структура диалога, выбор сценария его развития, семантика и синтаксис сообщений. Рассматриваются модели построения интерфейса, типы и структуры окон, основные компоненты графического интерфейса, предоставляющие возможность изменять содержимое или форму представления отображаемой информации, а также управлять работой приложений. Приведены цели, задачи, и виды юзабилити-тестирования, а также основные организации, оказывающие влияние на процесс стандартизации технологий, описаны примеры стандартов и необходимость их использования.

Для слушателей специальности 1-40 01 74 «Web-дизайн и компьютерная графика» заочной формы обучения.

УДК 004.738.12(075.8)
ББК 32.81я73

Содержание

ВВЕДЕНИЕ В ПРЕДМЕТ. ОСНОВНЫЕ ПОНЯТИЯ

Основные понятия

Цель данного курса состоит в изучении основных идей и проблем в сфере проектирования пользовательских интерфейсов программных средств, принципов и стандартов интерфейсов, методик оценки качества интерфейсов программного обеспечения (ПО).

Центральными понятиями курса являются термины юзабилити и информационная архитектура. Неразрывно с ними связаны такие понятия, как «человеко-машинное взаимодействие», интерфейс, эргономика, пользователь.

Определим и рассмотрим подробнее перечисленные понятия.

Информационная архитектура (Information architecture (IA)) – это совокупность методов и приемов организации информации для облегчения выполнения пользователями их информационных запросов по поиску и просмотру информации.

Юзабилити – это свойство системы или продукта. «Юзабельность» продукта означает, что пользователь может достичь своих целей при использовании этого продукта. Специалисты по юзабилити должны учитывать в своей работе эти цели.

Определения юзабилити согласно стандарту ISO 9241-11 звучит так: «*юзабилити* – это степень эффективности, продуктивности и удовлетворенности, с которыми продукт может быть использован определенными пользователями в определенном контексте использования для достижения определенных целей».

Научной основой для юзабилити является *человеко-машинное взаимодействие* (Human-computer interaction (HCI)). Это научная и прикладная дисциплина, предметом которой является то, как пользователи используют компьютеры и как следует разрабатывать компьютерные системы, для того, чтобы обеспечить более эффективное их использование этими пользователями. Она включает

в себя некоторые элементы эргономики, психологии, графического дизайна, информатики, социологии и антропологии.

Факторы, значимые для HCI:

- эргономичность;
- модели пользователя (восприятие, моторика, мышление, взаимодействие, организация работы, адаптация к многообразию);
- окружение HCI (средства взаимодействия, гипермедиа и Web, средства связи);
- разработка и развитие систем, ориентированных на пользователя;
- критерии и проверка легкости использования;
- принципы разработки удобных пользовательских HCI.

Эргономика (от греч. érgon – работа и nómos – закон) – научная дисциплина, комплексно изучающая человека (группу людей) в конкретных условиях его (их) деятельности в современном производстве.

Юзабилити – это раздел эргономики, посвящённый разработке ПО. Некоторые исследователи считают, что юзабилити и эргономика это отдельные понятия, поскольку эргономика делает упор на физиологическом удобстве, а юзабилити в большей степени рассматривает психологические аспекты.

В последнее время термин «юзабилити» используется как синоним слова «эргономика» в контексте таких продуктов, как бытовая электроника или средства связи.

Основное правило юзабилити состоит в том, что *пользователь всегда прав. И если он в чем-то ошибается, то это не его трудности, а проблемы системы* (Рольф Молич, датский юзабилити-специалист).

Информационная архитектура

Проектирование IA сайта состоит из анализа его содержания и разработки структуры, которая будет содействовать пользователю при решении его информационных задач (поиск товара, поиска ответа на вопрос и т.д.).

Хорошо разработанная структура упрощает создание навигационной системы и макетов страниц, поэтому качественная IA – это синоним хорошего юзабилити. Полезность методов юзабилити будет тратиться впустую на поддержание фундамента каждой страницы сайта без эффективной IA. Слово «архитектура» используется здесь по одной очень важной причине: информационный архитектор создает чертежи, на которых потом создается весь сайт.

Большинство веб-дизайнеров при разработке сайта основное внимание уделяют тому, как он выглядит. Содержимое сайта, его структура, навигация, названия пунктов меню, подписи к элементам форм остаются на втором плане. Сайт с недостаточно спланированной структурой может помешать пользователям в решении их задач. Чтобы этого избежать, содержимое сайта должно иметь понятную логическую структуру. Только в этом случае сайт станет эффективным.

Разработка качественной IA не простое занятие. Пользователи не заинтересованы в том, чтобы разбираться со структурой сайта только для того, чтобы достичь своих целей. Если они могут решать свои задачи без осмысления структуры сайта, значит спроектированная IA работает. Таким образом, IA должна быть невидима пользователю, но в то же время она должна быть логичной и интуитивной. Вот почему ее так сложно создать и так легко использовать.

При создании IA необходимо учитывать различные типы поведения пользователей, которые возникают при поиске

информации. Существуют два основных типа: просмотр и поиск. Пользователь, который предпочитают просматривать информацию, классифицированную по категориям, будет разочарован, если не сможет найти на сайте каталог, и наоборот, пользователь, предпочитающий поиск, придет в замешательство, если на главной странице не окажется формы поиска. Поэтому в структуру сайта необходимо закладывать различные способы организации информации для разных типов поведения пользователей. Кроме разработки иерархической структуры сайта, информационный архитектор должен создать его навигационную схему. При этом необходимо руководствоваться следующими вопросами: какие элементы будут принадлежать глобальной навигации (пункты меню, которые находятся на всех страницах сайта), какие элементы будут локальными (меню, которое показывается в подразделах сайта), а какие элементы будут зависеть от контекста страницы (ссылки, которые показываются внутри конкретной страницы).

Для создания структуры сайта и навигации используются таксономии. *Таксономия* – это система классификации, в которой информация группируется по категориям (элемент может входить только в одну категорию). Архитектор начинает создавать структуру меню (глобального и локального) и навигацию только после того, как информация распределена по таксономиям.

Классификация информации на категории дает возможность продумать смысловое значение элементов информации для пользователя. Специфические термины, жаргон могут внести осложнения для восприятия, понимания и, как следствие, повлиять на выбор пользователем различных пунктов меню. Эти термины необходимо определить во время разработки IA и заменить на те, которые будут понятны пользователю. Например, можно разработать словарь и включить в него базовые термины, используемые на сайте, что придаст информации целостность и связность.

После определения таксономий, структуры, навигации и подписей элементов меню, можно приступить к разработке макета для того, чтобы отобразить, как все эти элементы IA будут выглядеть на сайте. После этого создается детальный графический дизайн сайта.

Вероятность того, что пользователь найдет необходимую информацию, намного выше для тех сайтов, для которых IA намеренно разрабатывалась по сравнению с сайтами, для которых IA получилась сама собой, без проработки.

Интерфейс

Интерфейс – это совокупность технических, программных и методических (протоколов, правил, соглашений) средств сопряжения в вычислительной системе пользователей с устройствами и программами, а также устройств с другими устройствами и программами.

Выделяют следующие виды интерфейса:

- интерфейс пользователя – набор методов взаимодействия компьютерной программы и пользователя этой программы;
- программный интерфейс – комплекс правил и соглашений для взаимодействия между программами;
- физический интерфейс – способ взаимодействия физических устройств.

Интерфейс пользователя – это совокупность средств, при помощи которых пользователь общается с различными устройствами, чаще всего – с компьютером или бытовой техникой.

Интерфейс пользователя компьютерного приложения включает:

- средства отображения информации, отображаемую информацию, форматы и коды;
- командные режимы, язык «пользователь-интерфейс»;

- устройства и технологии ввода данных;
- диалоги, взаимодействие и транзакции между пользователем и компьютером, обратную связь с пользователем;
- поддержку принятия решений в конкретной предметной области;
- порядок использования программы и документацию на неё.

Графический интерфейс пользователя (Graphical User Interface (GUI)) в вычислительной технике – это система средств для взаимодействия пользователя с ПК, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана – окон, значков, меню, кнопок, списков и т.п. (например, ОС Windows). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или мыши) ко всем видимым экранным объектам.

Впервые концепция GUI была предложена учеными из исследовательской лаборатории Xerox PARC в 1970-х, но получила коммерческое воплощение лишь в продуктах корпорации Apple Computer. В настоящее время GUI является стандартной составляющей большинства доступных на рынке операционных систем и приложений.

Интерфейс командный – это вид диалогового взаимодействия пользователя с ПК, при котором используются различные команды, набираемые на клавиатуре и отображаемые на экране.

Интерфейс командной строки – это разновидность консольного интерфейса человека и компьютера, в котором инструкции компьютеру даются только путём ввода с клавиатуры текстовых строк – команд (например, ОС MS DOS). Формат вывода информации в интерфейсе командной строки не регламентируется; обычно это

также простой текстовый вывод, но может быть и графическим, звуковым и т.д.

Интерфейс многооконный – это вид диалогового взаимодействия пользователя с ПК, при котором каждой программе или данным отводится своя прямоугольная область (окно) на экране.

Сам термин «человеко-компьютерное взаимодействие» говорит о наличии человека в качестве одной из сторон взаимодействия. Каждый интерфейс предназначен для использования одной или несколькими конкретными категориями пользователей, которые обладают определенными характеристиками. Не существует интерфейсов, которые были бы одинаково удобны в использовании для всех людей. Поэтому для дизайнера интерфейсов уже на начальных этапах разработки необходимо учитывать, кто будет пользоваться системой, которую он проектирует, и в каких условиях это будет происходить.

Таким образом, проектирование интерфейса не является частью процесса разработки ПО, а должно быть частью процесса создания спецификаций на систему.

Любой интерфейс, независимо от сферы его применения, имеет пять основных характеристик:

1) производительность пользователей;
2) количество человеческих ошибок;
3) скорость обучения работе с системой;
4) субъективное удовлетворение пользователей;
5) способность сохранения пользователями навыков работы с системой в течение длительного времени.

Последняя характеристика не важна относительно систем автоматизации, поскольку считается, что работники пользуются системой достаточно часто, поэтому потери навыков произойти не может. Значительно испортить систему могут низкие показатели по всем остальным характеристикам.

Веб-дизайнер

В последнее время особое внимание уделяется качеству разрабатываемых систем. Поэтому все чаще среди объявлений о поиске персонала можно встретить вакансию веб-дизайнера. Рассмотрим основные обязанности и функции веб-дизайнера.

- *Исследование деятельности пользователей системы* (формализация контекста использования системы, ролей пользователей; проведение исследований, наблюдений, анкетирования, интервью с конечными пользователями; определение целей и задач пользователей, а также целей и задачи системы).
- *Проектирование общей структуры системы и разработка навигации* (проведение проектирования с точки зрения пользователя, а не разработчика).
- *Детальное проектирование экранов системы*.
- *Планирование, проведение и анализ результатов юзабилити-тестирования системы*.
- *Участие в разработке документации* (составление стилевых руководств для проектируемой системы; разработка технического задания и пользовательской документации).

Одно из основных отличий программиста от веб-дизайнера состоит в том, что программист работает с машинами, а дизайнер интерфейсов – с людьми. Веб-дизайнер должен уметь применять методы качественных социологических исследований, проводить наблюдения (например, за работой пользователей в естественной обстановке), разбираться в социальной и когнитивной психологии, психологии труда. Требовать наличия всех этих знаний и умений от квалифицированного программиста не стоит, поскольку у него техническое образование, кроме того, вся его профессиональная

деятельность имеет слабое отношение к работе с потребностями и слабостями людей.

Процесс разработки информационной архитектуры всегда сталкивается со многими проблемами, связанными с ограничениями бюджета, временными рамками проекта, определенными техническими ограничениями и т.д. Часто приходится искать компромисс между эстетикой и эргономикой. Поэтому редко можно встретить систему, которая была бы одновременно привлекательна и эргономична. Приемлемый результат невозможен без совместной работы дизайнера и разработчика интерфейсов.

ИСТОРИЯ ВОЗНИКНОВЕНИЯ ЮЗАБИЛИТИ И HCI

Человеко-компьютерное взаимодействие (HCI), как отдельная дисциплина, начала развиваться в конце 70-х – начале 80-годов. Толчком к ее развитию послужило осознание проблем, которые стали возникать у пользователей при работе со сложными и мощными компьютерными система.

Каждый день мы становимся свидетелями или участниками изменений в качестве и удобстве пользовательских интерфейсов. Ранее разработка визуального интерфейса велась ближе к концу проекта. Кроме того, хоть этим процессом и занимались опытные программисты, но они имели смутные (или совсем не имели) представления о способностях пользователей.

Когнитивные науки и их практические выводы позволили изучить возможности пользователей компьютерных систем. В ответ на появление таких знаний возникла дисциплина HCI, которая использует их в разработке пользовательских интерфейсов.

1940-е – Эргономика

Задолго до возникновения HCI было известно, что причины многих интерфейсных проблем связаны с человеческой психологией.

Во время 2-й мировой войны британскими психологами были проведены исследования внимания операторов радарных установок. В результате были разработаны рекомендации по оптимизации дизайна мониторов радарных установок, благодаря которым операторы могли концентрировать внимание на мониторах и различать сигналы даже в состоянии усталости или задумчивости.

К концу 40-х годов произошло формирование основных эмпирических знаний о человеческой деятельности и, в частности, о психологических аспектах профессиональной деятельности человека. Результатом этого стало возникновение формальной дисциплины о человеческих рабочих системах – эргономики.

1950-60-е – Возникновение компьютеров

В течение одного десятилетия компьютеры стали доступными. Несмотря на то, что компьютеры были большими, дорогими и очень редкими, они смогли произвести революцию в способе работы людей.

В 1960 году была запущена первая компьютеризированная система обработки транзакций. Это была система резервирования авиабилетов «Sabre». В том же году в Массачусетском университете (США) была создана программа, которая могла сдать на отлично университетский минимум по дифференциальному исчислению.

Новые возможности, которые открылись благодаря компьютерам, казались безграничными. К сожалению, первые системы, подобные «Sabre», обладали существенным недостатком: огромные размеры и сложности устройства не позволяли большинству людей разобраться с ними. Возникшие проблемы взаимодействия компьютеров и людей на рабочих местах позволили Д.Р. Ликлидеру в 1960 году создать концепцию соединения человеческого мозга и компьютерных машин для обработки информации, которую в рамках эргономики стали называть человеко-машинным интерфейсом. Под термином «*MMI design*» (Man-Machine

Interface Design – проектирование человеко-машинных интерфейсов) стали понимать применение известных принципов эргономики к проектированию интерфейсов с целью достижения наилучшей связи человека и компьютера.

В 60-е года был заложен фундамент революции 80-90-х годов в сфере персональных компьютеров и Интернета. Возникли новые изобретения, которые сегодня всем нам известны: Дуг Енглбарт (1962) изобрел мышь, а Тед Нельсон (1960) придумал термин «Гипертекст». В 1969 году Министерство Обороны США поручило четырем ведущим университетам провести исследование сетей. Заказ состоял в создании системы проводной связи, которая функционировала бы даже в условиях уничтожения более 30% линий. Сетевой принцип связи был изобретен именно этими университетами. Так появился ARPANET, предшественник современного Интернета.

1970-е – Развитие ПК

В 70-х годах вычислительная мощность компьютеров выросла и упала в цене. Впервые компьютеры стали появляться на рабочих столах у людей. После изобретения персональных компьютеров (ПК) компания IBM удивила многих людей тем, что разделила «железо» и ПО и разрешила другим программам работать на своих ПК.

Компании-производители ПО стали получать выгоду от такого разделения, но в то же время стали испытывать трудности с технической поддержкой собственного ПО в связи с ее высокой стоимостью. Исправить ситуацию можно было путем разработки продуктивных и эффективных интерфейсов, которые бы решали задачи пользователей без их обращения в службу техподдержки.

1980-е – GUI

К концу 70-х годов многие исследователи искали альтернативы интерфейсу командной строки. В проект Xerox «Star», который был

открыт в 1981 году, были заложены основы того, что Бен Шнайдермен позже (1982) назвал «прямым манипулированием». Графическое взаимодействие имело преимущества перед интерфейсом командной строки: инкрементные действия (например, копирование файла требовало одновременно указать место хранения, команду на копирование и место куда, сейчас интерфейс сообщает о возможных действиях) и быстрая обратная связь от интерфейса, возможность отменять действия и использование наглядных действий вместо «изучения» нового языка (командной строки).

В проекте Xerox «Star» был использован стандарт WIMP (Windows, Icons, Menus, Pointers – окна, иконки, меню и указатели). Впервые система для ПК была создана на основе методов, которые в дальнейшем стали универсальными методами юзабилити: прототипирование и анализ, тестирование с пользователями и итеративная детализация элементов интерфейса. К сожалению, этот проект потерпел неудачу, в основном из-за того, что основными пользователями ПК тогда являлись бизнесмены, а в «Star» не хватало некоторых ключевых функций электронных таблиц. Таким образом, «Star» нарушил один очень важный принцип юзабилити: *одного удобства – мало, продукт должен быть еще и полезным*.

В дальнейшем новый графический интерфейс (GUI) был освоен несколькими компаниями (Lisa, VisiCalc – 1983, Apple Macintosh – 1984, Microsoft Windows, Commodore Amiga – 1985). Но на протяжении 80-х годов люди оставались равнодушными к GUI в виду отсутствия программ, использующих возможности нового интерфейса, большого разнообразия конкурирующих платформ и недостатка вычислительных мощностей. Распространение GUI началось после выпуска ОС Windows 3.0 компанией Microsoft на фоне увеличения количества ПК.

Многие инновации в графических интерфейсах имеют корни в академических исследованиях 60-70-х годов. К 80-м годам эти исследования значительно отклонились от классической эргономики:

игнорировались физические аспекты рабочих систем. Внимание было направлено на когнитивные аспекты взаимодействия человека и компьютера. Термин «человеко-компьютерное взаимодействие» стал использоваться вместо устаревшего (и политически некорректного) «человеко-машинного интерфейса».

1990-е – Интернет

К середине 90-х самой распространенной операционной системой стала Windows, что привело к появлению стандартов интерфейса. Многие новые интерфейсы просто внешне подражали стилю Windows. Интерфейс в этих продуктах часто нарушал стандарты и не был интуитивно понятен пользователю.

Возможность соединять компьютеры посредством сети позволила работать над проектами и общаться посредством чата и электронной почты на любом расстоянии. Социальные аспекты работы с компьютерами стали частью разработки успешных систем. Вместо моделирования взаимодействия одного пользователя с компьютером, проектировщики начали моделировать взаимодействие группы людей посредством компьютера. Все это создало спрос на юзабилити-специалистов в командах разработчиков ПО.

Дисциплина HCI, основанная на психологии, позволяла экспертам моделировать сложные социальные взаимодействия, окружающие новую вычислительную технику. Эти модели позволили исследователям разработать рекомендации и правила по созданию веб-интерфейсов.

Сама дисциплина HCI не осталась без развития. Основанная на моделях обработки когнитивной психологии, HCI стала использовать такие отрасли знания, как этнографию, лингвистику, теорию коммуникаций и гуманитарные науки.

2000-е – Мобильные технологии и ближайшее будущее

Многим разработчикам казалось, что они знают потребности и желания своих пользователей, но они даже не задумывались об использовании методов разработки, ориентированных на нужды пользователей. Плохое качество моделей привело к тому, что многие компании выбросили деньги на создание продуктов, которые никто не мог и не хотел использовать. Компании, которые смогли предоставить полезные и удобные сервисы, до сих пор существуют на рынке.

Методы юзабилити и HCI доказали свою полезную роль в предотвращении рисков проектирования. Кроме использования для веб-сервисов, эти методы также используются в проектировании мобильных систем. С конца 90-х годов мобильные устройства расширили способы, с помощью которых мы взаимодействуем с компьютерами и друг с другом. Мобильные телефоны, PDA (Personal digital appliances) и беспроводные сети дали начало новой концепции вычислительной техники: мир, в котором технология – повсюду. Люди перестали быть пользователями одиночных, изолированных устройств. Очевидно, что системы, которые имеют такое влияние на способы нашего функционирования, не могут разрабатываться при помощи старых технологий. Системы, предназначенные для людей, должны быть сконцентрированы на них при проектировании, и это задача для юзабилити на ближайшее время.

ВИЗУАЛЬНЫЕ АТРИБУТЫ ОТОБРАЖЕНИЯ ИНФОРМАЦИИ

Композиция и организация экрана

К визуальным атрибутам отображаемой информации относятся:

- взаимное расположение и размер отображаемых объектов;
- цветовая палитра;
- средства привлечения внимания пользователя.

Проектирование размещения данных на экране предполагает выполнение следующих действий:

1) определение состава информации, которая должна появляться на экране;
2) выбор формата представления информации;
3) определение взаимного расположения данных или объектов на экране;
4) выбор средств привлечения внимания пользователя;
5) разработка макета размещения данных на экране;
6) оценка эффективности размещения информации.

Процесс проектирования необходимо повторять до тех пор, пока разработчик и потенциальные пользователи не будут удовлетворены.

Общие принципы расположения информации на экране должны обеспечить для пользователя:

- просмотр экрана в логической последовательности;
- простоту выбора нужной информации;
- идентификацию связанных групп информации;
- различимость сообщений об ошибках или предупреждений;
- возможность определить, какое действие со стороны пользователя требуется (и требуется ли вообще) для продолжения выполнения задания.

В зависимости от специфики выполняемого пользователем задания, решается какая информация подлежит отображению. Существенную роль здесь играет правильное разбиение задания на этапы, не требующие одновременного присутствия большого объема данных на экране. Это условие вытекает из ограниченности

кратковременной памяти человека, способной хранить одновременно не более пяти-девяти объектов. Если вся информация исходного документа не помещается на одном экране, некоторые элементы данных можно повторить на других экранах для сохранения целостности и последовательности обработки, причем их расположение должно оставаться прежним.

Не следует заставлять пользователя дополнительно обрабатывать информацию, например, уточнять по справочникам значения кодов, производить какие-либо преобразования, пересчеты и т.п. Форматы вывода даты, времени и других подобных стандартизированных данных должны быть общепринятыми, а не индивидуальными для данной системы. Восприятию текста способствует общепринятая система сочетания в нем больших и малых букв.

Эффективное восприятие информации также определяется рациональным размещением данных на экране. Существуют некоторые правила, регулирующие плотность расположения данных на экране (или в пределах окна):

- оставлять пустым приблизительно половину экрана (окна);
- оставлять пустую строку после каждой пятой строки таблицы;
- оставлять четыре-пять пробелов между столбцами таблицы.

Данные необходимо располагать на экране так, чтобы взгляд пользователя сам перемещался в нужном направлении. Содержимое полей не должно «прижиматься» к краю экрана, а располагаться около его горизонтальных или вертикальных осей. Меню, содержащее относительно небольшой объем информации, необходимо смещать в левую верхнюю часть экрана. Содержимое и наименования полей, относящихся к одной группе, нужно выравнивать по вертикали для сохранения симметрии. По

возможности необходимо выравнивать все логически связанные группы данных.

При расположении данных также следует учитывать право-левую асимметрию головного мозга человека: при запоминании образов ведущую роль играет правое полушарие, а при запоминании слов более активно левое. Информация с правой части экрана поступает непосредственно в левое полушарие, а с левой части – в правое. В связи с этим рекомендуется группировать текстовые сообщения справа, а изображения – слева. У некоторых людей это распределение функций полушарий противоположно. У женщин асимметрия выражена слабее, чем у мужчин.

Учет асимметрии памяти имеет существенное значение, если интервалы следования сообщений не превышают 10 с. Поэтому приведенные рекомендации в первую очередь следует учитывать в интерфейсах программ, работающих в режиме реального времени.

Методы выделения информации

Рациональное размещение данных на экране – не единственный метод обеспечения удобства пользовательского интерфейса. Современные мониторы предоставляют различные методы выделения выводимой информации на экране, т.е. использование таких атрибутов, которые позволяют привлечь внимание пользователя к некоторой области экрана. В этом качестве можно использовать цвет символов, цвет фона, уровень яркости, мерцание и применение различных шрифтов для выводимых символов. Часто для выделения информации используется подчеркивание, вывод в инверсном виде, различные рамки и тени. Однако основной рекомендацией по их применению является использование минимально необходимого числа атрибутов.

Качество визуального проектирования в значительной степени влияет и на психофизиологическое состояние пользователя, и на

эффективность его работы в целом. Наиболее всего это проявляется при использовании GUI. Несмотря на то, что внешний вид приложения во многом определяется его предназначением, для всех GUI-приложений справедливы следующие положения:

- все графические элементы приложения создают единую визуальную среду;
- каждый графический элемент и реализуемая им функция тесно взаимосвязаны, и эта связь должна быть интуитивно понятна пользователю.

Таким образом, при проектировании интерфейса приложения необходимо учесть такие факторы, как эффективное использование пространства экрана, выбор формы представления объектов, цветовая палитра и композиция графических элементов, а также выбор средств привлечения внимания пользователя к тем или иным элементам информации, отображаемой на экране.

Композиция и организация

Человек лучше воспринимает визуальную информацию, если она соответствующим образом организована в пространственном отношении. В первую очередь взгляд привлекают цветные элементы, а не черно-белые; графические, а не текстовые; изолированные (отдельно стоящие), а не сгруппированные.

Поэтому при проектировании визуальных элементов интерфейса приложения целесообразно опираться на определенные принципы:

1) иерархическая организация отображаемой информации;
2) визуальное выделение наиболее важных элементов;
3) сбалансированность структуры экрана;
4) визуальное объединение логически взаимосвязанных элементов;

5) удобочитаемость и логическая согласованность отображаемой информации;
6) интеграция (использование единых подходов к визуализации отображаемой информации не только в рамках приложения, но и рабочей среды в целом).

Принцип иерархической организации информации означает размещение информации с учетом ее значения относительно других визуальных элементов приложения. Результат упорядочения влияет на реализацию остальных принципов. Для успешной реализации этого принципа необходимо получить ответы на следующие вопросы:

- какая информация наиболее важна для пользователя?
- что пользователь должен видеть на экране в первую, во вторую, в третью очередь?
- что пользователь хочет или должен делать в первую, во вторую, в третью очередь?

При реализации принципа визуального выделения наиболее важных элементов необходимо решить следующие задачи: выбор на каждом шаге работы некоторой основной идеи, наиболее важной для выполнения этого шага; соответствующее представление и размещение реализующих эту идею элементов.

В силу указанных выше психофизиологических особенностей, в первую очередь человек обращает внимание на верхний левый угол просматриваемой области или на ту ее часть, которая визуально отличается от других. Поэтому наиболее важную информацию лучше размещать либо в верхнем левом углу экрана, либо в окне, снабженном специальными атрибутами.

Принцип сбалансированности структуры экрана предполагает, с одной стороны, рациональное использование пространства экрана, а с другой – такое размещение информации, при котором на экране в

каждый момент времени представлена только та ее часть, которая действительно необходима для выполнения очередного шага задания пользователя.

Принцип визуального объединения логически взаимосвязанных элементов можно рассматривать как развитие предыдущего принципа. Объединение способствует уяснению пользователем того, как именно представленная на экране информация и как элементы управления связаны с выполняемым шагом задания и друг с другом. Например, если в диалоговой панели имеется кнопка, которая влияет на содержимое списка, целесообразно поместить их рядом.

Под принципом удобочитаемости и логической согласованности отображаемой информации понимается то, что любая информация (не только текстовая) должна быть выражена в компактной и доступной форме. Пользователь также должен быть способен уяснить, как она связана с предыдущим и последующими шагами задания. При реализации данного принципа полезно получить ответы на следующие вопросы:

- можно ли представленную на экране идею или понятие выразить более просто?
- легко ли пользователь может выполнить данный шаг?
- действительно ли все отображаемые элементы необходимы для выполнения этого шага?

Интеграция. Если интерфейс приложения визуально согласуется с интерфейсом системы и интерфейсом других приложений, значительно легче обеспечить пользователю последовательную и предсказуемую рабочую среду. Для того, чтобы осуществить этот принцип, необходимо получить ответы на следующие вопросы:

- насколько оказывается согласованным экранное представление различных компонентов приложения в процессе работы пользователя?

- как соотносятся визуальные параметры приложения с системным интерфейсом и интерфейсом других приложений?

Шрифт

Шрифты, как и остальные визуальные элементы, способствуют организации информации и созданию определенного настроения. Изменяя размер и плотность шрифта, можно указать пользователю на степень важности той или иной информации и порядок, в котором она должна быть прочитана.

Шрифты называют также «гарнитурами шрифтов». Они используются для вывода «глифов» (чисел, букв и других символов), которые группируют по семействам (все связанные), стилю (курсив, нормальный, наклонный, и т.д.), варианту (нормальный или малый прописной), толщине (жирность), растяжению (уменьшенный или увеличенный тип по высоте или ширине) и размеру (по высоте или ширине в пунктах или пикселях).

Одним из первых шагов по окончательному оформлению веб-дизайна является выбор шрифтов, которые будут использоваться на сайте. Множество гарнитур шрифта на сайте может запутать пользователя. С другой стороны, сайт, который использует везде один шрифт, кажется безвкусным. Выходом может послужить использование одного шрифта для заголовков и другого – для основного текста на всем сайте, что добавит сайту целостность, и позволит различать заголовки и основной текст при просмотре страницы.

Существует четыре основных типа шрифтов.

1) *Шрифт с засечками* (*Serif*): любая гарнитура шрифта, которая содержит завершающие штрихи, расширяющиеся или сужающиеся концы, или имеет реально отсеченные окончания

(включая прямоугольные засечки). Шрифт с засечками был выбран для печати основного текста, так как его легко читать на печатной странице. Но веб-страница отличается от печати, и некоторые исследования показывают, что иногда шрифты без засечек читать легче, чем такие как шрифт, показанный на следующем рисунке, когда он используется для основного текста на странице.

Times New Roman, regular, 18 point

2) *Без засечек* (*Sans-serif*): любая гарнитура шрифта, конечные штрихи которой не имеют никаких расширений, пересекающих штрихов, или других украшений.

Verdana, regular, 18 point

3) *Рукописный* или *курсив*: обычно похожи на буквы, написанные пером или кистью. Недостатком использования таких шрифтов на веб-странице, особенно в основном тексте, является трудность чтения больших объемов. Кроме того, не все браузеры одинаково выводят один и тот же шрифт.

Staccato222BT, regular, 18 point

4) *Специальные шрифты*, включая *моноширинный*: единственным критерием моноширинного шрифта является единая фиксированная ширина всех глифов, подобно тому, как может выглядеть машинописная страница. Моноширинные шрифты нашли свое применение на сайте при выводе программного кода, так как такой шрифт четко показывает каждую букву и символ, используемые в коде.

Приведенные иллюстрации шрифтов показывают, что не все типы шрифта создаются одинаковыми, даже если создаются одного размера в пунктах. Размер в пунктах определяет высоту букв, и некоторые шрифты будут больше при 18 пунктах, чем другие. Существуют также и другие различия, такие как расстояние между буквами и словами, или тот факт, что некоторые гарнитуры шрифтов, такие как Jokewood, не имеют букв нижнего регистра. Такие шрифты могут найти применение в рекламных объявлениях.

Одинаковые шрифты по разному могут выглядеть в различных браузерах, поскольку не все операционные системы поддерживают одни и те же шрифты. В связи с этим для вывода страницы можно использовать Serif Sans-serif.

Заказчик веб-страницы обычно не знает разницы между гарнитурами шрифта с засечками и без засечек. Следовательно, значение имеет разборчивость текста. Для этого необходимо убедиться, что:

- используемый шрифт имеет достаточно большой размер для чтения с различными разрешениями браузера;
- задан достаточный контраст между фоном и основным текстом;
- заголовки отличаются от основного текста;
- основной текст не растянут на всю ширину экрана;
- заголовки и основной текст написаны без ошибок.

«Многомерность» экрана

Повышению функциональности способствует использование перспективы, подсветки и затенения при изображении различных

элементов интерфейса с целью обеспечения эффекта трехмерного образа. Например, кнопку можно изобразить так, чтобы при ее выборе пользователь увидел, что кнопка действительно нажата. При создании эффектов полагают, что источник света находится в верхнем левом углу экрана.

При разработке визуальных элементов следует помнить о том, что изображение любого «трехмерного» объекта занимает на экране больше места, чем его «плоский» аналог. Например, трехмерные элементы можно использовать только для изображения интерактивных элементов. При этом трехмерными делать лишь те детали, которые действительно необходимы для идентификации образа пользователем.

Пространственное размещение и группирование

Размер и взаимное расположение визуальных элементов влияют на создание последовательной и предсказуемой среды. Визуальная структура важна также с точки зрения передачи назначения элементов, отображенных в окне.

Группирование представляет собой компактное размещение взаимосвязанных элементов. Для ее реализации можно использовать либо группирующий блок, либо просто разместить элементы на соответствующем расстоянии друг от друга.

Использование цвета для визуальной группировки объектов не всегда удачно, поскольку это может привести к изменению цветовой схемы.

С целью обеспечения единого подхода к пространственному размещению графических элементов введена специальная единица измерения – дискрета окна.

Дискрета окна (Dialog base unit – единица площади диалоговой панели) – это аппаратно-независимая величина; в горизонтальном направлении равняется одной четверти средней величины символов

текущего системного шрифта; по-вертикали – одной восьмой средней высоты символов текущего системного шрифта.

Рекомендуется оставлять промежуток между элементами группы, равный четырем дискретам, а расстояние до края окна и между группами должно быть не менее семи дискрет. Элементы управления в панели инструментов располагать так, чтобы от края панели до края окна оставался промежуток, равный ширине рамки окна. Между элементами самой панели должно быть не менее четырех дискрет высоты (если не группируется набор связанных кнопок). В некоторых случаях, например, когда кнопки панели инструментов используются подобно набору переключателей, они могут располагаться слитно (без промежутка).

Основные кнопки управления вторичного окна целесообразно сгруппировать в верхнем правом углу окна или расположить в виде линейки вдоль нижнего края окна.

Кнопки <OK> и <Отменить> должны располагаться рядом. Замыкать группу должна кнопка <Справка> (если она поддерживается приложением). Если кнопка <OK> не используется в данном окне, но имеются другие кнопки управления, то лучше всего установить кнопку <Отменить> в конце набора кнопок управления, но перед кнопкой <Справка>.

Выравнивание

Выравнивание так же можно рассматривать как способ визуального отображения взаимосвязанной информации (или элементов управления).

Различают три основных способа выравнивания информации:

- вертикальное (по левому или правому краю выравниваемых элементов);

- горизонтальное (по верхней строке или по верхнему краю элемента);
- смежное выравнивание (когда элементы смыкаются краями).

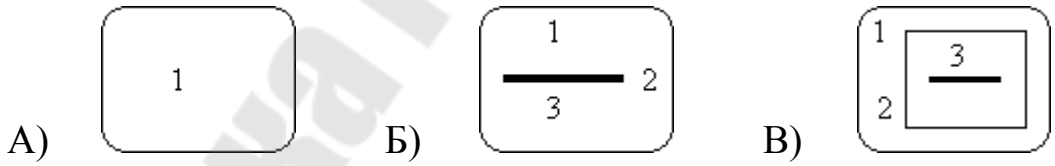
Если информация расположена вертикально, рекомендуется выровнять ее элементы по левому краю соответствующей области. Это облегчает пользователю быстрый просмотр информации.

Если в виде столбца выводятся числовые данные, значения которых могут изменяться, их лучше выровнять по правому краю.

Визуальные объекты

Назначение любого интерфейса – обеспечение взаимосвязи между пользователем и компьютером. Каждая строка, каждый управляющий элемент, блок, часть текста, цвет, рисунок, появляющийся на экране, – все это оказывает влияние на пользователя, как по отдельности, так и в комбинации.

Рассмотрим примеры.



Экран А) не имеет видимых элементов. Однако это не так – фон, хотя и пуст, является визуальным элементом.

Экран Б) содержит линию, проходящую по его центру. На этом экране три визуальных элемента: сама линия и области над и под ней.

На экране В) изображен блок, содержащий линию. Количество визуальных элементов, различаемых пользователем, равняется пяти: область вне блока, блок, линия и области внутри блока над и под линией.

С добавлением на экран новых элементов визуальное воздействие на пользователя усиливается. Полная симметричность всех объектов затрудняет чтение информации с экрана. Элементы одинакового цвета и размера воспринимаются как принадлежащие к одной группе. Управляющие элементы необходимо четко выравнивать. Объекты окна нужно создавать по образу реальных объектов.

РАЗРАБОТКА СТРУКТУРЫ ДИАЛОГА

Обмен информацией между пользователем и компьютером (точнее, его ПО) по всем формальным признакам соответствует понятию «диалог». Для того чтобы диалог был конструктивным, должны соблюдаться следующие правила:

- участники диалога должны понимать язык друг друга;
- участники диалога не должны говорить одновременно;
- очередное высказывание должно учитывать как общий контекст диалога, так и последнюю информацию, полученную от собеседника.

Если собеседники обсуждают вопросы, относящиеся к какой-либо специальной области, они должны придерживаться единой терминологии; при объяснении следует сначала пояснить основные термины и понятия.

Очень краткие ответы и слишком большие паузы могут сбить собеседника с толку, а злоупотребление специальными терминами или жаргонизмами вообще может привести к преждевременному завершению беседы.

Перечисленные выше факторы существенно влияют на структуру диалога, т.е. на форму общения.

Таким образом, при проектировании диалога, необходимо определить:

- структуру диалога;
- возможный сценарий развития диалога;
- содержание управляющих сообщений и данных, которыми могут обмениваться человек и приложение (семантику сообщений);
- визуальные атрибуты отображаемой информации (синтаксис сообщений).

Выбор структуры диалога

Рассмотренные ниже четыре варианта структуры диалога являются разновидностями структуры типа «вопрос-ответ», тем не менее, каждая из них имеет свои особенности и наиболее удобна для определенного класса задач.

Диалог типа «вопрос-ответ». Структура диалога типа «вопрос-ответ» (Q&A) основана на аналогии с обычным интервью. Система берет на себя роль интервьюера и получает информацию от пользователя в виде ответов на вопросы. Это наиболее известная структура диалога; все диалоги, управляемые компьютером, в той или иной степени состоят из вопросов, на которые пользователь отвечает, однако в структуре Q&A этот процесс выражен явно.

В каждой точке диалога система выводит в качестве подсказки один вопрос, на который пользователь дает один ответ. В зависимости от полученного ответа система может решить, какой следующий вопрос задавать.

Структура Q&A предоставляет естественный механизм ввода как управляющих сообщений (команд), так и данных. Никаких ограничений на диапазон или тип входных данных, которые могут обрабатываться, не накладывается.

Эта структура может удовлетворить требования различных пользователей и типов данных. Такая структура особенно уместна при реализации диалога с множеством ответвлений (в тех случаях, когда на каждый вопрос предусматривается большое число ответов, каждый из которых влияет на то, какой вопрос будет задан следующим). По этой причине, структура Q&A часто используется в экспертных системах.

Данная структура имеет ряд существенных недостатков. Во-первых, не гарантирует минимального объема ввода, оцениваемого по количеству нажатий клавиш; во-вторых, возможны проблемы с анализом и интерпретацией вводимых данных; в-третьих, процедура ввода ответов набором их с клавиатуры достаточно утомительна для пользователя.

Диалог на основе меню. Меню является наиболее популярным вариантом организации запросов на ввод данных во время диалога, управляемого компьютером.

Существует несколько основных форматов представления меню на экране:

- список объектов, выбираемых прямым указанием, либо указанием номера (или мнемонического кода);
- меню в виде блока данных;
- меню в виде строки данных;
- меню в виде пиктограмм.

Пользователь диалогового меню может выбрать нужный пункт, вводя текстовую строку, которая идентифицирует этот пункт, указывая на него непосредственно или просматривая список и выбирая из него. Система может выводить пункты меню последовательно, при этом пользователь выбирает нужный ему пункт нажатием клавиши.

Меню в виде строки данных может появляться вверху или внизу экрана и часто остается в этой позиции на протяжении всего диалога. Таким образом, посредством меню удобно отображать возможные варианты данных для ввода, доступных в любое время работы с системой.

Если в системе имеется достаточно большое разнообразие вариантов действий, организуется иерархическая структура из соответствующих меню.

Дополнительные меню в виде блоков данных «всплывают» на экране в позиции, определяемой текущим положением указателя, либо «выпадают» непосредственно из строки меню верхнего уровня. Эти меню исчезают после выбора варианта.

Меню в виде пиктограмм представляет собой множество объектов выбора, разбросанных по всему экрану; часто объекты выбора содержат графическое представление вариантов работы.

Меню можно с равным успехом применять для ввода как управляющих сообщений, так и данных. Приемлемая структура меню зависит от его размера и организации, от способа выбора пунктов меню и реальной потребности пользователя в поддержке со стороны меню.

Структура типа меню является наиболее естественным механизмом для работы с устройствами указания и выбора: меню представляет собой изображение тех объектов, которые выбираются пользователем. Если диалог состоит исключительно из меню, можно реализовать последовательный интерфейс, при котором пользователь применяет только устройства для указания; однако такое постоянство редко достижимо на практике. Следует отметить, что хотя работа с этими устройствами не требует профессионального владения клавиатурой, для подготовленного пользователя это не самый быстрый способ выбора из меню. Вместо указания пользователь может сообщить о своем выборе вводом соответствующего идентификатора.

Меню – это наиболее удобная структура диалога для неподготовленных пользователей; жесткая очередность открытия и иерархическая вложенность меню может вызывать раздражение профессионала, замедлять его работу. Традиционная структура меню недостаточно гибка и не в полной мере согласуется с методами адаптации диалога, такими, например, как опережающий ввод, с помощью которого можно ускорить темп работы подготовленного пользователя.

Диалог на основе экранных форм. Как структура типа «вопрос-ответ», так и структура типа меню предполагают обработку на каждом шаге диалога единственного ответа. Диалог на основе экранных форм допускает обработку на одном шаге диалога нескольких ответов.

На практике формы используются там, где учет какой-либо деятельности требует ввода стандартного набора данных. Человек работает с формой до тех пор, пока не заполнит ее полностью и не передаст системе. Система может проверять каждый ответ непосредственно при вводе или по окончании заполнения всей формы.

Сообщения об ошибках, выводимые непосредственно после ответа, могут отвлечь внимание, но могут оказать и положительное влияние. В тех случаях, когда информация для ввода выбирается из некоторого целостного документа, проверку лучше отложить до конца заполнения формы, чтобы не прерывать процесс ввода; если же такой целостности нет, то проверку следует выполнять сразу после ввода ответа (после заполнения очередного поля).

Если встретилась какая-либо ошибка, приложение не должно заново выводить пустую форму; выводится форма с предыдущими ответами и допущенными ошибками. Новый «бланк» выдается лишь в случае соответствующего запроса пользователя.

Такую структуру уместно применять там, где источником данных служит существующая входная («бумажная») форма документа.

Не обязательно, чтобы внешний вид этих форм совпадал, но все вводимые элементы данных должны располагаться в том же относительном порядке и иметь такой же формат, что и в исходном документе.

Часто все необходимые единицы ввода нельзя отобразить одновременно в пределах одного экрана (или окна), и их необходимо разделить на группы, которые отображаются на последовательности экранов (окон). Важно, чтобы это разбиение сохраняло логические связи и не приводило к разделению связанных частей документа.

Структура диалога на основе экранной формы обеспечивает высокий уровень поддержки пользователя: для каждой формы могут быть предусмотрены сообщения об ошибках и справочная информация. Пользователю можно также оказать помощь, включив некоторые элементы формата ответа в вопрос или в поле ответа.

Эта структура позволяет повысить скорость ввода данных по сравнению со структурой типа «вопрос-ответ» и манипулировать более широким диапазоном входных данных, нежели меню; кроме того, с ней могут работать пользователи любой квалификации.

Поскольку эта структура имеет последовательную, а не древовидную организацию, она в меньшей степени подходит для работы в режиме выбора вариантов. Еще одной областью применения экранных форм является задание параметров запроса в базах данных. Этот механизм иногда называют запросом по образцу (Query by Example).

Одним из типов заполнения форм являются также многовариантные меню. В таких меню пользователю предоставляется список вариантов, и он не ограничен возможностью единственного выбора; можно указать несколько вариантов.

Диалог на основе командного языка. Структура диалога на основе командного языка столь же распространена, что и структура типа меню. Она очень часто используется в операционных системах. Исторически это первая из реализованных структур диалога.

При такой организации диалога система не выводит ничего, кроме постоянной подсказки (приглашения на ввод команды), которая означает готовность системы к работе. Каждую команду вводят с новой строки и обычно заканчивают нажатием клавиши «ввод». Ответственность за правильность задаваемых команд ложится на пользователя. Система информирует о невозможности выполнения неверной команды, не поясняя, как правило, причин.

Подобно меню, диалог на базе команд удобен для ввода управляющих сообщений, однако он обеспечивает более широкие возможности выбора в любой точке диалога и не требует иерархической организации обслуживающих его программ.

Программная система может поддерживать достаточно большое количество команд, но на практике следует ограничивать их число, чтобы не перегружать память пользователя.

Структура на базе командного языка не отличается хорошей поддержкой пользователя и пригодна в основном для подготовленных специалистов. Более того, поскольку системе неизвестно, что намеревается делать пользователь, трудно представить какую-либо реальную помощь в процессе работы, кроме выдачи справок общего характера.

Поскольку данная структура предполагает большой объем запоминаемого материала, имена команд следует выбирать так, чтобы они несли смысловую нагрузку и легко запоминались.

Диалог должен управлять данными. В интерфейсах на основе командного языка это достигается с помощью составных командных строк, где команда предшествует списку параметров (входным данным). Параметры в списке можно задавать в одной из двух форм – позиционной и ключевой.

Назначение позиционного параметра определяется по его месту в командной строке. Позиционные параметры уменьшают объем вводимой информации, но их существенным недостатком является то, что вводимые значения должны указываться в строго определенном порядке, нарушение которого может повлечь непредсказуемые последствия. Задание позиционных параметров осложняется, если их список достаточно велик. Этот недостаток стремятся компенсировать путем пропуска неизменяемых параметров, вводя два разделителя друг за другом.

В случае ключевых параметров каждое значение предваряется определенным идентификатором, который определяет его назначение. Ключевые параметры уменьшают нагрузку на память пользователя в том отношении, что отпадает необходимость в запоминании порядка их следования; можно опускать необязательные параметры. Недостатком является то, что пользователю приходится запоминать множество ключевых слов, а разработчику – подбирать для них «осмысленные» имена. Этот подход требует большего времени работы системы, чтобы распознать ключевые слова, заданные в произвольном порядке.

Структура на основе языка команд по своим возможностям самая быстрая и гибкая из всех структур диалога. Большинство пользовательских интерфейсов на базе естественного языка реализуется с помощью языков команд с очень большим набором ключевых слов.

Подготовленный пользователь испытывает удовольствие от ощущения того, что он управляет системой, а не наоборот. Однако эта структура не обеспечивает пользователя поддержкой, поэтому даже подготовленные пользователи считают, что очень сложно использовать все заложенные в ней возможности. Большинство пользователей хорошо знакомы только с весьма ограниченным набором средств, с которыми работают регулярно.

Разработка сценария диалога

Развитие диалога во времени можно рассматривать как последовательность переходов системы из одного состояния в другое. Ни одно из этих состояний не должно быть тупиковым, т.е. пользователь должен иметь возможность перейти из любого текущего состояния диалога в требуемое (за один или несколько шагов). Для этого в ходе разработки интерфейса необходимо определить все возможные состояния диалога и пути перехода из одного состояния в другое – разработать сценарий диалога.

Цели разработки сценария:

- выявление и устранение возможных тупиковых ситуаций в ходе развития диалога;
- выбор рациональных путей перехода из одного состояния диалога в другое (из текущего в требуемое);
- выявление неоднозначных ситуаций, требующих оказания дополнительной помощи пользователю.

Сложность разработки сценария определяется в основном двумя факторами – функциональными возможностями создаваемого приложения (т.е. числом и сложностью реализуемых функций обработки информации) и степенью неопределенности возможных действий пользователя.

Степень неопределенности действий пользователя зависит от выбранной структуры диалога. Наибольшей детерминированностью обладает диалог на основе меню, наименьшей – диалог типа «вопрос-ответ», управляемый пользователем.

Таким образом, сценарий диалога можно упростить, снизив степень неопределенности действий пользователя. Возможные способы решения этой задачи:

- использование смешанной структуры диалога (применение меню с целью «ограничения свободы» пользователя там, где это возможно);
- применение входного контроля вводимой информации (команд и данных).

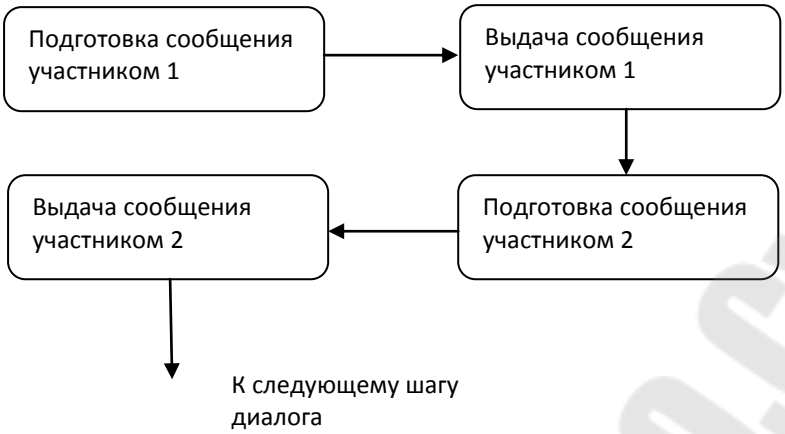
Дополнительные возможности по снижению неопределенности действий пользователя предоставляет объектно-ориентированный подход к разработке интерфейса, при котором для каждого объекта заранее устанавливается перечень свойств и допустимых операций. Наиболее эффективен такой подход при создании графического интерфейса.

Сокращая число возможных состояний диалога, разработчик должен помнить о необходимости отражения в его сценарии работы средств поддержки пользователя (что усложняет сценарий). Способ описания сценария диалога зависит от степени его сложности. Методы описания сценариев делятся на две группы: неформальные и формальные методы.

Главное достоинство формальных методов состоит в том, что они позволяют автоматизировать как проектирование диалога, так и его модификацию (адаптацию) в соответствии с характеристиками пользователя.

В настоящее время наиболее широко используются формальные методы описания сценариев на основе сетей Петри и их расширений, а также на основе систем представления знаний (фреймовые модели и продукционные системы).

Независимо от способа описания сценария его основной структурной единицей является шаг диалога, соответствующий одному акту взаимодействия пользователя с системой.



Сценарий диалога позволяет описать процесс взаимодействия пользователя с приложением на уровне решаемой им прикладной задачи. Однако для программной реализации интерфейса такое описание носит слишком общий характер. Поэтому на этапе реализации необходимо перейти на уровень описания соответствующих процессов с помощью используемых инструментальных средств разработки приложения.

Описание структуры диалога с помощью сети переходов

Развитие диалога можно рассматривать как последовательность переходов от одного состояния к другому. Диалог может находиться в особом состоянии ожидания ввода от пользователя и будет переходить в одно из нескольких возможных состояний в зависимости от характера принятой информации. В соответствии с этим диалог можно представить в виде сети переходов или диаграммы состояний. Каждое состояние представляется вершиной графа, помеченной соответствующим ей номером. Будем рассматривать вершину как некоторую точку, в которой диалог выводит сообщение пользователю или требует входного сообщения от пользователя. Связи между вершинами обозначаются направленными дугами, соединяющими две вершины, метка на дуге определяет условие, при выполнении которого возможен переход.

Может существовать несколько дуг, соединяющих две вершины и определяющих, что переход может быть вызван несколькими условиями.

Выделяют три типа вершин.

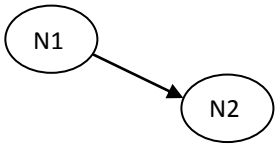
1. Вершина, в которой выводится сообщение пользователю с запросом на ввод. Передача на соседнюю вершину зависит от контекста введенного сообщения.
2. Вершина, в которой выводится сообщение пользователю без запроса на входное сообщение; следует автоматическая передача на соседнюю вершину.
3. Вершина, в которой выводится сообщение пользователю с запросом на ввод, после которого осуществляется безусловный переход на соседнюю вершину.

Рассмотрим различия между вершинами типа 2 и 3. В типе 2 переход осуществляется автоматически, т.е. не требуется никакого входного сообщения. В типе 3 перехода не произойдет до тех пор, пока не будет сделан ввод, но, независимо от контекста введенных данных, будет иметь место один и тот же переход.

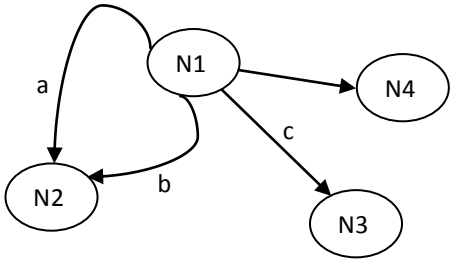
Каждая вершина в сети переходов представляет собой отдельное состояние диалога. Она образует точку переключения в развитии диалога. Переключением можно управлять придерживаясь структуры, в которой устанавливается соответствие между назначением следующей вершины и каждым элементом из набора условий.

Условие	**Следующая вершина**
C(1)	N(1)
C(2)	N(2)
…	…
C(k)	N(k)
	N(k+1)

Каждому условию C(i) соответствует вершина N(i), в которую должен быть осуществлен переход. Если никакое из условий не выполнено, то переход произойдет в вершину N(k+1) (например, в случае ошибки). Такое же обозначение используется и для безусловного перехода. Рассмотрим примеры.



Условие	Следующая вершина
	N2



Условие	Следующая вершина
a	N2
b	N2
c	N3
	N4

ОСУЩЕСТВЛЕНИЕ ДИАЛОГА С ПОМОЩЬЮ ПРОЦЕССОВ ВВОДА-ВЫВОДА

Термин «процесс» используется для описания последовательности операций, выполняемых системой.

Термин «задание» – для обозначения того, что хочет сделать пользователь. Например, задание по вычислению средней прибыли группы товаров. Теоретически существует бесконечное множество заданий, которые нужно выполнить; на практике большинство систем имеет дело лишь с ограниченным числом процессов

Каждое задание может быть выполнено одним или несколькими процессами. Не существует однозначного соответствия между заданиями и процессами. Один процесс может использоваться для выполнения нескольких заданий; одно задание может выполняться несколькими процессами.

При разработке интерфейсов процессы рассматриваются как «черные ящики». В общем случае их называют процессами по выполнению задания или просто заданиями.

Интерфейс человек-компьютер обеспечивает связь между пользователем и процессом, выполняющим задание. Это дает возможность пользователю определять, какие задания сделать активными в данный момент, как передавать им данные для обработки и принимать результаты обработки. С точки зрения программного обеспечения в состав интерфейса входят два компонента: набор процессов ввода-вывода и процесс диалога.

Пользователь компьютерной системы взаимодействует с интерфейсом: через интерфейс он посылает входные данные и принимает выходные. Процессы по выполнению заданий вызываются интерфейсом в требуемые моменты времени.

Процессы ввода-вывода

Процессы ввода-вывода служат для того, чтобы принять от пользователя и передать ему данные через различные физические устройства. При выборе устройств учитываются следующие факторы.

Содержание и формат обрабатываемых данных. Для некоторых прикладных задач нужен лишь ограниченный диапазон текстовых символов, для других – графический режим с высокой разрешающей способностью. Иногда пользователь должен вводить набор произвольных величин, а иногда нужно сделать только выборку из небольшого набора возможных значений.

Объем ввода-вывода. Увеличение объема входных данных предполагает наличие косвенного механизма ввода, например автоматического сбора данных.

Ограничения, накладываемые пользователем и рабочей средой (физические дефекты пользователя, работа в цеху, совмещение работы за компьютером с другими видами работ).

Ограничения, связанные с другими аппаратными и программными средствами, которые используются в системе.

С каждым устройством связан свой процесс ввода-вывода, задача которого – воспринять данные от пользователя и преобразовать их во внутреннее представление, с которым может работать процесс диалога.

Очевидно, что процесс ввода с клавиатуры и процесс речевого ввода различаются типами обрабатываемых ими устройств. Отделяя физический процесс ввода-вывода от процесса диалога, можно добиться того, что смена устройств ввода-вывода не приведет к изменению процесса диалога, а сведется лишь к замене процесса ввода-вывода.

Процессы ввода-вывода обеспечивают обмен информацией на самом верхнем уровне процесса диалога. На этом уровне диалоговый процесс должен правильно интерпретировать каждое слово и даже каждый звук.

Задачи диалогового процесса:

- определение задания, которое пользователь возлагает на систему;
- прием логически связанных входных данных от пользователя и размещение их в переменных соответствующего процесса в нужном формате;
- вызов процесса выполнения требуемого задания;
- вывод результатов обработки по окончании процесса в подходящем для пользователя формате.

Сообщения

В диалоге информация передается в виде сообщений. В любом диалоге существует несколько типов сообщений: команда, данные – при вводе; подсказка, данные, состояние, ошибка, справка – при выводе.

Подсказка – это выходное сообщение системы, побуждающее пользователя вводить данные.

Реакция пользователя на подсказку может вызвать процесс выполнения задания или какую-нибудь функцию диалогового процесса – *входное управляющее сообщение*; или передать процессу выполнения задания входные данные – такой тип сообщений называется *входными данными*. Могут быть и сложные сообщения, которые за один сеанс ввода вызывают нужный процесс и вводят данные.

Обычно для диалогового процесса нужно проверять введенные пользователем данные на наличие ошибок. Характер проверки зависит от формата входного сообщения.

Управляющее сообщение проверяется на совпадение с одним из элементов списка возможных значений.

Входные данные проверяются на нахождение их в пределах допустимого диапазона.

Сообщение об ошибке – это сигнал диалогового процесса о том, что невозможно дальнейшее выполнение работы, т.к. вызванный процесс выполнения задания не может обработать сообщение, введенное пользователем.

Введенное пользователем сообщение преобразуется диалоговым процессом в стандартный формат и передается на вход другого процесса. Это задание возвращает диалоговому процессу либо подтверждение о получении входных данных, либо результат обработки. Диалоговый процесс, в свою очередь, преобразует это

выходное сообщение в подходящий для пользователя формат и выводит его в виде данных или как сообщение о состоянии системы.

Выходные данные – это данные, которые возвращает диалоговый процесс по окончании обработки.

Сообщение о состоянии системы – это информация для пользователя о том, что произошло или происходит в системе.

Справочная информация – выводится в тех случаях, когда пользователь не может ответить на запрос системы, потому что ему непонятен запрос или он забыл, что нужно вводить. Справочная информация поясняет, что делать дальше и почему.

Входные сообщения

Диалог можно классифицировать с учетом формата входных сообщений и гибкости, позволяющей пользователю вводить входные сообщения, когда ему удобно. Входное сообщение позволяет:

- выбрать режимы диалога, например, получение справки;
- выбрать нужный процесс выполнения задания;
- ввести данные для выполнения задания.

Диалог, управляемый системой – это диалог, в котором процесс жестко задает, какое задание можно выбрать и какие данные вводить.

Диалог, управляемый пользователем – это диалог, в котором инициатива принадлежит пользователю, т.е. он непосредственно подает команду на выполнение нужного на данном этапе задания.

Диалог, управляемый системой, более удобен, потому что он лучше подстраивается под пользователя, но при этом имеет больше ограничений, чем диалог, управляемый пользователем. Предполагается также, что задания структурированы (обычно иерархически) и диалог ведется в соответствии с этой структурой.

Формат, в соответствии с которым пользователь вводит свои сообщения, называют *грамматикой диалога* (коды, цепочки ключевых слов).

Подсказки

Существует ряд форматов вывода подсказок в диалоге.

Самый сложный формат – *меню*, когда наряду с запросом на ввод сообщения выводятся допустимые форматы ввода.

Выводить меню в виде текста необязательно. Можно вывести пиктограммы различных возможностей. Формат меню правомерен в том случае, если диапазон правильных входных данных не слишком велик и может быть явно выведен на экран. С помощью меню обычно выводятся команды системы.

Система может с помощью *вопроса* уточнить, какой тип данных требуется. Подсказка может содержать указание на требуемый формат входного сообщения.

Если требуется ввести данные сложного формата, например, даты, можно использовать в качестве подсказки специальную *форму* ввода.

Еще один вид подсказки – *запрос на ввод команды* (данных).

Независимо от грамматики или способа ведения диалога, в его основе лежит следующий цикл:

- явный или неявный запрос на ввод данных;
- ввод данных через процесс ввода;
- проверка входных данных.

Этот цикл повторяется, пока не будут приняты приемлемые входные данные.

Если выводится запрос на ввод команды, следующий шаг обработки будет зависеть от введенной команды.

Ввод-вывод информации

Классифицируем сообщения по характеру информации:

- Вывод текстового сообщения:
  - в текущую позицию на устройстве;
  - в заданную позицию на устройстве;
  - с заданием конкретного формата отображения.
- Ввод текстового сообщения:
  - с использованием стандартных процедур ввода;
  - в режиме посимвольного ввода;
  - с использованием управляющих символов.
- Ввод сообщений типа «Указать» и «Выбрать»:
  - просмотр предлагаемого списка операций и выбор нужной;
  - выбор данных из любого места экрана.
- Вывод графического сообщения.
- Ввод графического сообщения.

Текстовое сообщение – это строка символов.

Графическое сообщение – это сообщение, которое нельзя представить в виде строки символов. Обычно описывается на битовом уровне.

Сообщения типа «Указать» и «Выбрать» – особый тип входных данных для обеспечения выбора из допустимого множества возможных альтернатив.

Управляющие сообщения обычно включают выбор из некоторого ограниченного множества вариантов. Входные данные содержат информацию о таком выборе.

Возможные альтернативы представляются списком выбираемых объектов. Такими объектами могут быть отдельные символы, строки текста, фрагменты экрана и т.д.

Каждый объект определяется следующими характеристиками.

Содержание – описывает объект; каждый объект должен однозначно определяться, чтобы пользователь знал, к чему обращаться.

Поле, в пределах которого объект отображается на экране; поля должны быть достаточно широкими и хорошо отделены друг от друга (всегда легче указать на большой объект, нежели на маленький).

Множество атрибутов, которые описывают его формат; область экрана, в которой находится объект, должна быть отчетливо выделена.

Число объектов не должно быть большим, чтобы пользователь не терялся при выборе. Существуют два основных способа указания пользователем определенного объекта.

Абсолютный выбор позволяет пользователю указать любое место на экране независимо от того, размещается там объект или нет. Соответствующий процесс ввода получает от устройства ввода точные координаты. Удобен при использовании мыши, светового пера, сенсорного экрана.

При относительном выборе пределы перемещения по экрану ограничены только списком объектов. Пользователь всегда будет выбирать объекты только из этого списка.

Для информировании пользователя о положении указателя выбора используются изменение атрибутов, графические указатели, перемещение курсора и т.п.

При выборе необходимо определить способ завершения процесса ввода, т.е. «фиксации» выбранного объекта. С этой целью часто используется нажатие клавиши ENTER, двойной щелчок мышью.

Методы разработки гибкого интерфейса

Гибкость интерфейса заключается в способности приложения адаптироваться (пользователем или автоматически) к любому возможному уровню подготовки пользователя. Существуют три вида адаптации: фиксированная, полная и косметическая.

При *фиксированной адаптации* пользователь явно выбирает уровень диалоговой поддержки. Простейший вариант такой адаптации основан на использовании правила двух уровней, согласно которому система обеспечивает два вида диалога: подробный (для начинающего пользователя); краткий (для подготовленного пользователя).

Правило двух уровней может быть расширено до правила N уровней диалога. Однако такой подход имеет несколько недостатков:

- не учитывается тот факт, что навыки накапливаются постепенно;
- пользователь может хорошо знать одну часть системы и совсем не знать другую;
- пользователь сам определяет уровень своей подготовки, что снижает объективность оценки.

При *полной адаптации* диалоговая система стремится построить модель пользователя, которая по мере обучения последнего и определяет стиль диалога в зависимости от этих изменений. При этом одной из основных проблем является распознавание характеристик пользователя. Для ее решения необходимо определить, что использовать в качестве таких характеристик: время, затрачиваемое пользователем на ответ, количество обращений за помощью или характер ошибок и тип запрашиваемой помощи.

В настоящее время полная (автоматическая) адаптация практически ни в одной диалоговой системе не реализована.

Косметическая адаптация призвана обеспечить гибкость диалога без учета поведения пользователя, но и без однозначного

выбора им конкретного стиля диалога. Такая адаптация может быть достигнута за счет применения следующих методов:

- использование умолчаний;
- использование сокращений;
- опережающий ввод ответов;
- многоуровневая помощь;
- многоязычность.

Использование умолчаний**.** Сущность умолчания состоит в том, что система использует некоторое изначально заданное значение какого-либо параметра, пока пользователь не изменит его. В этом случае имеют место два аспекта адаптации системы: во-первых, начинающий пользователь имеет возможность использовать большинство параметров системы по умолчанию; во-вторых, система может запоминать значения, либо заданные при последнем сеансе работы, либо наиболее часто используемые.

Для удобства начинающих пользователей значения, используемые по умолчанию, могут выводиться на экран вместе с соответствующим вопросом системы, например: «Дата регистрации документа? [текущая]».

Самый распространенный способ принятия значений по умолчанию – это нулевой ввод, т.е. просто нажатие клавиши ENTER в качестве ответа на вопрос системы. Если используется командный язык, то пользователь просто пропускает параметр, используемый по умолчанию.

Использование сокращений предполагает, что пользователь вместо полного имени команды может вводить ее любое допустимое сокращенное обозначение. На первый взгляд может показаться, что сокращенный ввод более удобен для начинающего пользователя. Но это не совсем так. Чтобы пользователь мог, не задумываясь, заменить

команду корректным сокращением, он должен достаточно хорошо представлять имеющийся набор команд, усвоить лексику системы.

Одной из модификаций этого подхода является *опережающий ввод символов*, при котором система, «узнав» по первым символам команду, «дописывает» ее сама.

Идея опережающего ввода ответов заключается в том, что пользователь имеет возможность на очередном шаге диалога вводить не один ответ, а цепочку последовательных ответов, упреждая возможные вопросы системы.

Один из методов обеспечения многоуровневой помощи состоит в том, что сначала на экран выводится сообщение начального уровня, а затем пользователь может уточнить полученную информацию, используя переход на более низкий уровень по ключевому слову. На таком принципе основана работа многих современных Help-систем, обучающих гипертекстовых систем.

Сущность многоязычности интерфейса состоит в том, что структура и семантика диалоговых сообщений, которые выдает и получает пользователь, должны отвечать нормам родного языка пользователя и не зависеть от того, на каком языке разработаны инструментальные средства, которые он использует.

Возможный подход к реализации многоязычности – создание средств реакции системы на действия пользователя (сообщения-запросы, подсказки, сообщения об ошибках) отдельно от синтаксиса языка программирования (инструментальных средств).

Темп ведения диалога

Темп ведения диалога зависит от характеристик аппаратных и программных средств, а также от специфики решаемых задач. Кроме того, темп ведения диалога зависит от психофизиологических характеристик пользователя.

Время ответа (отклика) системы определяется как интервал между событием и реакцией системы на него. Данная характеристика интерфейса определяет задержку в работе пользователя при переходе к выполнению следующего шага задания.

Требования к времени ответа зависят от того, что ожидает пользователь от работы системы, и от того, как взаимодействие с системой влияет на выполнение заданий. Исследования показали, что если время ответа меньше ожидаемого, точность выбора операции из меню увеличивается с увеличением времени ответа системы. Это связано с тем, что излишне быстрый ответ системы как бы подгоняет пользователя, заставляет его суетиться в стремлении не отставать от более расторопного партнера по общению.

Время ответа должно соответствовать естественному ритму работы пользователя. В обычном разговоре люди ожидают ответа около 2 секунд и ждут того же при работе с компьютером. Время ожидания зависит от их состояния и намерений. На представления пользователя оказывает сильное влияние также его предшествующий опыт работы с системой.

Обычно человек может одновременно запомнить сведения о пяти-девяти предметах. Считается также, что хранение данных в кратковременной памяти ограничено по времени: около 2 с. для речевой информации и 30 с. для сенсорной. Поэтому люди разбивают свою деятельность на этапы, соответствующие порциям информации, которые они могут хранить одновременно в памяти.

Завершение очередного этапа называется *клаузой*. Задержки, препятствующие наступлению клаузы, очень вредны и неприятны, т.к. содержимое кратковременной памяти требует постоянного обновления и легко стирается под влиянием внешних факторов. Зато после клаузы подобные задержки вполне приемлемы и даже необходимы.

Завершение задачи, ведущее к отдыху, называют *закрытием*. В этот момент исчезает необходимость дальнейшего хранения

информации и человек получает существенное психологическое облегчение.

Так как пользователи интуитивно стремятся к закрытию в своей работе, следует делить диалоги на фрагменты, чтобы пользователь мог вовремя забывать промежуточную информацию.

Пользователи, особенно новички, обычно предпочитают много мелких операций одной большой операции, т.к. в этом случае они могут не только лучше контролировать общее продвижение решения и обеспечить его удовлетворительный ход, но и отвлечься от деталей работы на предыдущих этапах.

Имеющиеся результаты исследований позволили выработать следующие рекомендации по допустимому времени ответа интерактивной системы:

- 0,1-0,2 с – для подтверждения физических действий (нажатие клавиши, работа со световым пером, мышью);
- 0,5-1 с – для ответа на простые команды (от момента ввода команды, выбора альтернативы из меню до появления нового изображения на экране);
- 1-2 с – при ведении связного диалога (когда пользователь воспринимает серию взаимосвязанных вопросов как одну порцию информации для формирования одного или нескольких ответов, задержка между следующими друг за другом вопросами не должна превышать указанную длительность);
- 2-4 с – для ответа на сложный запрос, состоящий в заполнении некоторой формы. Если задержка не влияет на другую работу пользователя, связанную с первой, могут быть приемлемы задержки до 10 с;
- более 10 с – при работе в мультизадачном режиме, когда пользователь воспринимает данную задачу как фоновый процесс. Принято считать, что если пользователь не получает ответ в течение 20 с, то это не интерактивная система. В таком случае пользователь

может забыть о задании, заняться решением другой задачи и возвращаться к нему тогда, когда ему будет удобно. При этом программа должна сообщать пользователю, что задержка ответа не является следствием выхода системы из строя (например, путем регулярного обновления строки состояния системы или ведения протокола выполнения задания пользователя).

МОДЕЛИ ИНТЕРФЕЙСА. ОКНА.

Модели построения интерфейса

В соответствии с концепциями, положенными в основу графического интерфейса, объекты приложения могут быть визуально представлены на Рабочем столе либо в виде пиктограмм, либо в виде окон, отображающих содержимое объекта.

Во многих случаях для реализации взаимодействия пользователя с объектами приложения или приложением в целом оказывается достаточным единственного первичного окна, возможно, дополненного набором вторичных окон.

Однооконная модель приложения облегчает пользователям ассоциативную связь между объектами и их визуальным представлением, существенно упрощает пользователю работу с окнами.

Объекты некоторых типов (например, устройства) могут не требовать создания первичного окна и использовать только вторичное окно для просмотра и редактирования их свойств. Иногда объект может быть представлен в приложении лишь своей пиктограммой.

При выполнении некоторых заданий однооконная модель не обеспечивает достаточно эффективного управления приложением или отдельными его объектами; такая ситуация может иметь место в тех случаях, когда пользователю требуется работать одновременно с несколькими различными форматами представления одних и тех же данных или с несколькими видами взаимосвязанных данных в

пределах одного окна. В таких случаях следует использовать другие модели приложения: на основе *многодокументного интерфейса* (MDI) или *проекта*.

Техника взаимодействия пользователя с приложением существенно зависит от выбранной модели его построения.

Пиктограммы

Все пиктограммы, используемые в приложении, следует разрабатывать как единый набор. При этом должна обеспечиваться их согласованность и друг с другом, и с заданиями пользователя.

Каждая пиктограмма должна быть реализована в трех стандартных форматах:

- 16x16 пикселов (для 16 цветов);
- 32x32 пиксела (для 16 цветов);
- 48x48 пикселов (для 256 цветов).

Для меньших размеров может быть использована большая глубина цвета, но это требует увеличения памяти для хранения пиктограмм и более жестких требований к конфигурации компьютера.

Система автоматически формирует цветовую схему пиктограммы для монохромных конфигураций. Тем не менее, следует заранее оценить качество зрительного восприятия разработанных пиктограмм в монохромном режиме. Если результат окажется неудовлетворительным, необходимо разработать собственные монохромные варианты пиктограмм.

Пиктограммы разрабатываются не только для исполнимого файла приложения, но и для всех типов файлов данных, поддерживаемых вашим приложением. При этом пиктограммы для файлов данных (или документов) должны отличаться от пиктограмм

приложения, но иметь с ними общий элемент. Пиктограммы должны отображать сущность хранимой информации.

Все созданные пиктограммы должны быть зарегистрированы в системном реестре, иначе система будет автоматически использовать вместо них системные пиктограммы.

В основу рисунка, отображаемого на пиктограмме, должен быть положен образ объекта реального мира, точнее, те его детали, которые действительно необходимы для однозначного восприятия объекта пользователем. Где это возможно, лучше использовать трехмерное изображение и светотень.

Выводимое на пиктограммах изображение должно вызывать у пользователя вполне определенную предсказуемую ассоциацию с объектами реального мира.

Виды и структура окон

Окна предоставляют доступ к различным видам информации и классифицируются согласно своему предназначению.

Первичное окно. Взаимодействие с объектами реализуются средствами первичного окна, в котором происходит первоначальный просмотр и редактирование данных.

Типовая структура первичного окна:

- рамка – определяет размеры окна;
- заголовок окна – идентифицирует информацию, представленную в окне, может содержать кнопки управления первичным окном (Закрыть, Развернуть/Восстановить, Свернуть);
- полосы прокрутки – используются, если объем выводимой информации превышает текущий размер окна;
- другие элементы интерфейса (меню, панель инструментов, строка состояния).

Внешний вид рамки окна определяется типом окна. Изменяемое окно имеет четкую границу, которая обеспечивает управление размерами на основе прямого манипулирования. Если окно не может изменять размеры, граница сливается с краем окна.

Первичное окно содержит уменьшенную копию пиктограммы объекта или приложения, к которому оно относится. Она выводится в левом верхнем углу окна – в полосе заголовка и выбирается по следующим правилам:

- если окно относится к компоненту приложения, не создающему свои файлы данных, то используется пиктограмма самого приложения;
- если приложение обеспечивает работу с документами (файлами) различных форматов, то используйте пиктограмму, соответствующую формату отображаемого в окне документа;
- если приложение использует многодокументный интерфейс, пометите пиктограмму приложения в заголовке родительского окна, а в заголовке дочернего окна – пиктограмму конкретного типа файла данных.

Как отмечалось выше, поле заголовка содержит кнопки управления первичным окном. Для первичных окон в число этих кнопок не включается кнопка для вызова контекстно-зависимой справочной информации. Если наличие справки необходимо, то соответствующая кнопка включается в панель инструментов.

Для кнопок управления первичным окном используются следующие правила:

- если команда не поддерживается окном – не отображайте соответствующую кнопку;
- кнопка закрытия окна всегда должна быть самой правой кнопкой. Оставляйте промежуток между ней и другими кнопками;
- кнопка Свернуть должна предшествовать кнопке развернуть;

• кнопка Восстановить всегда заменяет кнопку Развернуть или кнопку Свернуть после выполнения соответствующей команды.

Использование подокон. Окно может разделяться на две и более относительно независимых областей, которые называются *подокнами*. Разделение окна позволяет, например, просматривать одновременно две части одного документа или отображать одну и ту же информацию в различной форме.

Если необходимо одновременно получить доступ к нескольким файлам при выполнении одного задания, следует использовать технологию MDI.

Разбиение окна на подокна может быть установлено либо разработчиком приложения как основная форма окна, либо пользователем, посредством задания соответствующего параметра.

Для того чтобы поддерживать разбиение окна, которое не определено заранее, включите в состав создаваемой программы *блок разделения*. Блок разделения – специальный элемент управления, который отображается в конце полосы прокрутки окна и обозначает регулируемую границу между подокнами.

Пользователь может изменять размеры подокон, перемещая блок разделения в нужную позицию.

При использовании подокон каждое из них должно иметь собственные значения атрибутов. При этом область выбора следует отображать только в активном подокне.

Когда основное окно закрывается пользователем, следует запоминать состояние подокон (количество, расположение, отображаемая информация, состояние выбора) как часть информации о состоянии этого окна. Это необходимо для восстановления окна.

Вторичные окна. Вторичные окна предназначены для приема от пользователя или отображения дополнительной информации об объектах, представленных в первичном окне. Они позволяют

устанавливать дополнительные параметры обработки или обеспечивают доступ к более специфическим деталям взаимодействия с объектами первичного окна.

Вторичные окна обладают некоторыми свойствами первичных окон, тем не менее отличаются от первичных во многих аспектах поведения и использования.

Для вторичных окон не создаются кнопки на панели задач!

Стандартное вторичное окно содержит:

- полосу заголовка окна;
- поле, ограниченное рамкой.

Пользователь может перемещать вторичное окно с помощью мыши. Нежелательно изменять его размеры, кроме окна палитры, поскольку любое вторичное окно предназначено для отображения конкретной предопределенной информации.

В некоторых случаях может возникнуть необходимость последовательного уточнения или дополнения отображаемой в окне информации; в таком окне может использоваться специальная кнопка – Дополнить.

Вторичное окно не имеет кнопок управления Развернуть и Свернуть. Для закрытия окна используется кнопка Закрыть.

Заголовок вторичного окна является его меткой и поясняет назначение окна; полоса заголовка вторичного окна не содержит пиктограммы.

Разрешается включать во вторичные окна строку состояния, но не рекомендуется дублировать в ней элементы, используемые в строке состояния первичного окна.

Вторичное окно может содержать в полосе заголовка окна кнопку вызова контекстной помощи (Что это?). Эта кнопка позволяет пользователю получать контекстно-зависимую справочную информацию о компонентах, отображенных в окне.

Вторичное окно может быть независимым или модальным.

Независимое вторичное окно позволяет пользователю взаимодействовать с другими вторичными или первичными окнами, а также переключаться между первичными окнами. Независимое вторичное окно целесообразно использовать в тех ситуациях, где пользователю может потребоваться повторить действие, связанное с этим окном (например, при поиске слова в тексте или при форматировании текста).

Модальное вторичное окно требует от пользователя завершить ввод данных в пределах данного окна и закрыть его, прежде чем продолжить работу за пределами окна. Вторичное окно может быть модальным по отношению к своему первичному окну или по отношению к системе. В последнем случае пользователь должен выполнить требующиеся действия и закрыть окно прежде, чем взаимодействовать с любыми другими объектами или окнами.

Модальные вторичные окна используются только в ситуациях определенного типа:

- когда для выполнения команды требуется ввести дополнительную информацию;
- когда необходимо приостановить работу пользователя, пока не будет выполнено некоторое условие.

Избегайте использования системных модальных вторичных окон, если ваше приложение не исполняется в качестве системной утилиты; но даже и в этом случае применяйте их только в наиболее серьезных ситуациях, игнорирование которых может привести к фатальной системной ошибке.

При выборе расположения вторичного окна на экране следует учитывать большое число факторов: назначение окна, причину его появления, размеры экрана и т.д.

Вторичное окно следует отображать в той позиции, где оно появлялось в последний раз.

При первом открытии окна установите его в позиции, удобной для работы пользователя (окно должно отображаться полностью!).

Удобно располагать вторичное окно таким образом, чтобы оно находилось в центре первичного окна по горизонтали и ниже заголовка окна, меню и всех панелей инструментов.

Виды вторичных окон:

1) Панель свойств – наиболее универсальное средство представления свойств объекта. Представляет собой независимое вторичное окно, которое отображает доступные пользователю свойства объекта, причем необязательно пользователю должно быть предоставлено право изменять их. Панель свойств отображается на экране по команде Свойства для конкретного (выбранного) объекта. Стандартные кнопки панели свойств – ОК, Отменить, Применить.

2) Панель контроля параметров. Реализуется в виде модального вторичного окна, связанного с тем объектом, свойства которого она отображает. Панель контроля параметров всегда относится к выбранному в данный момент объекту. При изменении параметров свойства применяются динамически – не требуется кнопок для подтверждения или отказа.

3) Диалоговые панели обеспечивают обмен информацией или ведение диалога между пользователем и приложением. Используются для получения от пользователей дополнительной информации, необходимой для выполнения команды или задания. Название диалоговой панели должно отражать название команды.

4) Окно Палитра – является независимым вторичным окном, которое содержит набор взаимосвязанных элементов управления. В виде такого окна может быть представлена панель инструментов или ее часть. Каждое окно Палитра может иметь собственное название, отображаемое в полосе заголовка, и собственный формат. Полоса заголовка содержит только одну кнопку Закрыть.

5) Окно Сообщение – предназначено для вывода на экран сообщений пользователю; обычно это информация о конкретной ситуации или условиях выполнения операций. Окна сообщений содержат графический символ, который указывает на тип выводимого сообщения, и собственно текст сообщения.

6) Всплывающие окна – используются для отображения дополнительной информации в тех случаях, когда в основном окне она представлена в сокращенной форме; для вывода контекстно-зависимой справочной информации. Всплывающая подсказка – разновидность всплывающего окна – используется для пояснений к элементам управления панели инструментов.

Многодокументный интерфейс

В процессе работы с одним и тем же приложением пользователю может потребоваться иметь на экране несколько открытых окон, содержащих информацию различных типов, либо представляющих собой разное изображение одних и тех же данных. Для создания таких окон и управления ими существует специальная технология – многодокументный интерфейс (MDI).

Техника MDI заключается в использовании одного первичного окна, называемого *родительским* окном, которое может содержать набор взаимосвязанных с ним *дочерних* окон. Каждое дочернее окно – это также первичное окно, единственным ограничением для которого является то, что оно может появиться только в пределах родительского окна. Родительское окно обеспечивает как визуальное, так и операционное пространство для своих дочерних окон. Например, на дочернее окно обычно распространяется область действия меню родительского окна и, возможно, других элементов его интерфейса (панели инструментов, строки состояния и т.д.). Их вид может изменяться, если необходимо отразить команды и атрибуты активного дочернего окна.

Вторичные окна, такие как диалоговые панели, окна сообщений или панели свойств, появляются на экране как результат тех или иных действий пользователя в родительском или дочернем окне. Эти окна должны активизироваться и отображаться в соответствии с общими соглашениями для вторичных окон, связанных с первичным окном, даже если они относятся к дочернему окну.

Заголовок родительского окна обычно содержит пиктограмму и имя приложения или объекта, который он представляет. Заголовок дочернего окна содержит пиктограмму, представляющую тип документа или файла данных, и имя файла. Как для родительского окна, так и для всех его дочерних окон должны поддерживаться всплывающие меню; перечень команд в таком меню соответствует первичному окну.

Пользователь может активизировать MDI-приложение, либо непосредственно открыв его, либо открыв документ того типа, который поддерживается этим приложением. Если MDI-приложение активизировано посредством открытия документа, сначала открывается родительское окно, а затем внутри его рабочей области – дочернее окно, отображающее выбранный документ. Для того, чтобы упростить пользователю открытие других документов, связанных с этим приложением, включите в его интерфейс диалоговую панель «Открыть».

В том случае, когда пользователь непосредственно открывает документ за пределами родительского окна приложения, то если родительское окно уже открыто, следует создать второй экземпляр приложения (еще одно родительское окно), а не окно документа в существующем родительском окне. Хотя открытие нового дочернего окна может быть более эффективным, его появление может нарушить среду задания, уже установленную в этом родительском окне.

Поскольку дочерние окна являются разновидностью первичных окон, при их закрытии следуют соглашениям, принятым для первичных окон. Когда пользователь закрывает дочернее окно, любые

несохраненные изменения должны быть обработаны в соответствии с общими соглашениями для всех первичных окон.

Приложение не должно разрешать пользователю закрыть дочернее окно, если это не позволит ему продолжить работу с приложением.

Когда пользователь закрывает родительское окно, закройте все его дочерние окна. Где возможно, сохраняйте состояние дочернего окна (размер и положение внутри родительского окна) и восстанавливайте это состояние, когда пользователь вновь открывает окно.

Технология MDI имеет свои ограничения:

- Хотя пользователь может запустить приложение, открыв документ, для того, чтобы работать с несколькими документами одновременно требуется использовать интерфейс приложения.
- Если открыто несколько файлов в пределах одного родительского окна, может быть нарушена согласованность связи между дочерними окнами и отображаемыми в них объектами (файлы не связаны между собой).
- Родительское окно в действительности не содержит объекты, представленные в дочерних окнах. Это не позволяет обеспечить эффективную непрерывную работу пользователя (после закрытия родительского окна созданная ранее рабочая среда не восстанавливается).

Перечисленные недостатки MDI могут быть преодолены за счет применения альтернативных средств, таких как Рабочие области, Рабочие книги и Проекты. Хотя эти средства реализуют однооконную модель интерфейса, тем не менее они обнаруживают целый ряд достоинств, присущих технологии MDI.

Рабочая область – контейнер. Основное отличие Рабочей области от MDI заключается в использовании концепции

объединения отображаемых объектов. Это означает, что объекты, отображаемые в Рабочей области, могут соответствовать файлам, содержащимся в одном и том же контейнере. Внешне же соответствующие им окна выглядят как дочерние окна, расположенные в пределах родительского окна.

Таким образом, концепция использования Рабочей области подобна концепции использования Рабочего стола, за исключением того, что она сама является объектом, который может быть представлен в виде пиктограммы и отображен в виде открытого окна. Чтобы окно объекта могло появиться в Рабочей области, сам объект должен входить в состав соответствующего контейнера.

Для рабочей области должна быть предусмотрена команда «Сохранить все» – для сохранения содержимого всех объектов области.

В настоящее время реализация механизма хранения объектов зависит от типа используемого контейнера.

Рабочая книга. Рабочая книга – это еще один альтернативный вариант управления формой представления отображаемых данных, в основе которого лежит метафора книги или записной книжки. В Рабочей книге различные формы данных представляются как отдельные разделы в пределах одного первичного окна.

В качестве средств навигации между разделами Рабочей книги могут использоваться этикетки вкладок. Каждый раздел представляет данные, которые могли бы быть отдельным документом. Рабочая книга лучше подходит для представления таких данных, которые могут быть определенным образом упорядочены и этот порядок имеет существенное значение.

Для рабочей книги действительны те же соглашения, что и обеспечивающие связь между родительскими и дочерними окнами.

Как и для Рабочей области, должна быть предусмотрена команда Сохранить все.

Проект. Проект реализует еще один альтернативный способ управления окном, который предусматривает возможность отображения в одном окне взаимосвязанных объектов, представленных их пиктограммами. В этом смысле Проект подобен каталогу: для выбранного объекта открывается окно, которое относится к тому же уровню, что и родительское окно. В результате, каждое дочернее окно проекта может также иметь собственную кнопку входа на панели задач.

В отличие от каталога, проект обеспечивает управление из родительского окна окнами входящих в него объектов (если открыт документ, пользователь может закрыть окно каталога; при закрытии проекта – закрываются все окна проекта).

Для различных окон проекта возможен различный набор элементов управления.

По аналогии с Рабочей областью и Рабочей книгой, Проект должен иметь команды для создания новых объектов, для их пересылки в Проект и из него, а также для сохранения любых изменений объектов, входящих в проект. Окно проекта должно содержать средства для работы с самим проектом как с объектом (в том числе для изменения его свойств).

ЭЛЕМЕНТЫ УПРАВЛЕНИЯ

Под элементами управления (Controls) обычно понимаются компоненты графического интерфейса, которые предоставляют пользователю возможность изменять содержимое или форму представления отображаемой информации, а также управлять работой приложения. К элементам управления относятся списки, полосы прокрутки, кнопки и т.д.

Каждый элемент управления имеет уникальный образ и обеспечивает определенную форму взаимодействия пользователя с приложением. Система также поддерживает возможность создания

собственных элементов управления. Определяя такие элементы, следует учитывать существующие системные соглашения, принятые для стандартных элементов управления.

Элементы управления обеспечивают обратную связь с пользователем. Для элементов управления характерны следующие состояния:

- нормальное – элемент управления не выбран;
- активное – если указатель мыши (курсор) находится в *горячей зоне* или фокусе ввода;
- состояние выбора – активный элемент управления выбран (запущено соответствующее элементу управления действие);
- «недоступно» – элемент управления недоступен на данном этапе работы.

Горячая зона определяет, будет ли элемент управления реагировать на указатель. Границы горячей зоны зависят от типа элемента. Для некоторых элементов, например, кнопок, горячая зона совпадает с видимой границей элемента. Для других горячая зона может включать графический символ элемента управления и относящуюся к нему текстовую область (это справедливо для флажков и переключателей).

Для большинства элементов управления система обеспечивает вывод текстовой подсказки. Подсказка помогает пользователю определить назначение данного элемента управления. Если элемент управления не имеет подсказки, ее можно реализовать в виде статической текстовой области или в виде всплывающей подсказки (Tooltip).

Поскольку некоторые элементы управления могут обеспечивать специфические способы взаимодействия пользователя с приложением, для них целесообразно создавать всплывающие меню (если элемент управления используется для передачи величины,

имеющей несколько возможных значений, или для доступа к контекстно-зависимой справочной информации). Такие меню создаются по стандартным правилам, но щелчок левой клавиши мыши на выбранном пункте не запускает действие, связанное с элементом управления. Т.е. всплывающее меню элемента управления позволяет пользователю определить, какие действия он реализует в текущей ситуации, но не позволяет непосредственно выполнить эти действия.

Меню

Меню́ (Menu) – элемент интерфейса пользователя, позволяющий выбрать одну (в простейшем случае) из нескольких перечисленных опций. Для этого предусмотрены два основных процесса: навигация и выбор. Навигация (перемещение) реализуется с помощью указателя мыши; клавишами управления курсором. Выбор реализуется двойным щелчком левой клавиши мыши или нажатием клавиши ENTER.

Главное меню окна (Main menu). Одна из наиболее распространенных форм меню – линейная последовательность команд или разделов. В таком виде выполнено главное меню окна, расположенное непосредственно под полосой заголовка первичного окна (так называемая полоса меню).

Полоса меню содержит названия пунктов меню, каждый из которых предоставляет доступ к выпадающему меню.

Содержание главного меню и связанных с ним выпадающих меню определяется функциональным предназначением приложения и контекстом выполняемого пользователя задания. Если установлена такая конфигурация окна, при которой главное меню не отображается, то необходимо использовать элементы управления, которые обеспечат доступ к тем же функциям приложения, что и меню.

Выпадающее меню (Pull-down menu). Отображается как панель с пунктами меню, расположенными в виде столбцов. Хотя система позволяет выводить пункты меню в несколько столбцов, но делать это нежелательно, т.к. это усложняет работу пользователя.

Перемещение по выпадающему меню с помощью клавиш управления курсором рекомендуется организовывать по кругу, т.е. если указатель стоит на последнем пункте меню и курсор перемещен вниз, то произойдет переход курсора на первый пункт, а с первого при перемещении вверх – на последний.

Всплывающие меню (Pop-up menu). Всплывающее меню предоставляет пользователю эффективный способ доступа к операциям над объектами. Отображается в текущей позиции (соответствующей положению указателя) – избавляет пользователя от необходимости перемещаться по экрану для выбора действия через меню или панель инструментов.

Всплывающее меню содержит команды, учитывающие специфику выбранного объекта или текущей ситуации, а следовательно сокращает число команд, среди которых пользователь должен сделать выбор.

Позволяют минимизировать объем отображаемой на экране информации, поскольку появляются по требованию пользователя.

Не следует использовать всплывающее меню в качестве единственного доступного средства выполнения каких-то действий. Команды всплывающего меню не должны дублировать содержимое одного из выпадающих меню.

Принципы расположения команд во всплывающем меню:

- первыми должны располагаться основные команды для работы с объектом (открыть, исполнить, печать), другие команды, поддерживаемые объектом (определяемые непосредственное его свойствами или текущим контекстом), и команда <*Что это?*>, если она поддерживается системой;

- во вторую группу включают команды, реализуемые через буфер обмена (вырезать, копировать, вставить);
- последними должны идти команды редактирования дополнительных атрибутов объекта, если таковые имеются.

Открывает всплывающее меню щелчок правой клавишей мыши на выбранном объекте. При этом объект изображается как выбранный. Отображается всплывающее меню так, чтобы его левый верхний угол совпадал с позицией указателя; однако если при этом меню выходит за пределы экрана, его положение должно быть скорректировано.

Если при открытом всплывающем меню пользователь щелкает правой клавишей мыши за пределами области выбора, то открытое ранее всплывающее меню должно быть закрыто и открыто меню, относящееся к новой области выбора.

Каскадные меню. Это подменю, на которое распадается пункт меню более высокого уровня. Визуально на наличие каскадного меню указывает треугольник, выводимый рядом с родительским пунктом меню.

Каскадное меню может использоваться для предоставления пользователю возможности дополнительного выбора и для отображения иерархически связанных объектов.

Использование каскадных меню усложняет интерфейс, поэтому старайтесь применять этот вид только в тех ситуациях, где они действительно необходимы; минимизируйте количество уровней (в идеале – единственное подменю); не используйте их для доступа к распространенным, часто используемым командам.

Взаимодействие пользователя с каскадным меню подобно выпадающему меню, но каскадное меню отображается после некоторой задержки.

Если после открытия каскадного меню пользователь перемещает указатель к другому пункту родительского меню, то каскадное меню после короткой задержки закрывается. Эта задержка позволяет пользователю перейти из родительского меню в соответствующее каскадное без нажатия кнопки мыши.

Требования к оформлению меню.

1) Если меню содержит большое количество пунктов, и они могут быть сгруппированы по некоторому признаку, следует разделить такие группы (стандартный разделитель – горизонтальная прямая линия).

2) Следует визуально выделять недоступные в данный момент пункты. Обычно их «обесцвечивают» или не отображают совсем. Предпочтителен первый способ.

3) Для недоступных выборов следует сохранять функции подсказки (что собой представляет данная команда и почему она недоступна).

4) Если в некоторой ситуации ни все пункты меню недоступны, то сделайте недоступным все меню в целом.

5) Если команда меню требует ввода дополнительной информации, ее имя должно сопровождаться многоточием (…).

6) Если пункт меню служит для включения/выключения параметра состояния объекта, то слева от пункта меню при включении параметра устанавливается маркер флажка, при выключении – маркер снимается (такие параметры называются взаимонезависимыми).

7) Для визуального отображения выбора взаимозависимых пунктов меню используется маркер переключателя, слева от выбранного пункта меню.

8) Если две команды меню реализуют альтернативные состояния, то после выбора одного из них его название можно заменить в меню на альтернативное.

9) Используйте уникальные имена для пунктов в пределах одного меню; пункты с тем же названием могут повторяться в других меню, чтобы представлять аналогичные действия.

Кнопки

Кнопкой (Button) 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 понятие кнопок многие элементы управления, которые как кнопки по большей части не воспринимаются. Нажатие на такую кнопку запускает какое-либо явное действие.

Размеры и поля. Чем больше кнопка, тем легче попасть в нее курсором. Это правило по мере сил всеми соблюдается, во всяком случае, кнопок размером 5 на 5 пикселей уже практически не встретишь. Однако помимо простоты нажатия на кнопку есть другая составляющая проблемы: пользователю должно быть трудно нажать не на ту кнопку.

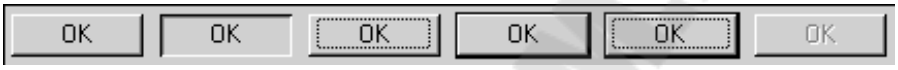
Добиться этого можно либо изменением состояния кнопки при наведении на неё курсором, либо установлением пустого промежутка между кнопками.

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

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.

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

Состояния. Кнопка должна как-то показывать пользователям свои возможные и текущие состояния. Количество состояний довольно велико, при этом наборы возможных состояний в ПО и в интернете значительно различаются. Например, кнопка в Windows может иметь шесть состояний: нейтральное, нажатое, нейтральное с установленным фокусом ввода, состояние кнопки по умолчанию, кнопка по умолчанию с установленным фокусом ввода и заблокированное состояние.



В интернете обычно используют меньший набор состояний: нейтральное, готовое к нажатию (onMouseOver) и активное (в случаях, когда набор кнопок используется для индикации навигации). Нажатое и заблокированное состояние используются очень редко, а «нейтральное с установленным фокусом ввода» старается, как может, создать браузер.

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

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

Флажки и переключатели

Флажок (Check box) и переключатель (Radio button) являются кнопками отложенного действия, т.е. их нажатие не должно инициировать какое-либо немедленное действие. С их помощью пользователи вводят параметры, которые скажутся после, когда действие будет запущено иными элементами управления. Нарушать это правило опасно, поскольку это серьезно нарушит сложившуюся ментальную модель пользователей. В этом заключается общность флажков и переключателей.

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

Из этого различия вытекают и все остальные. Например, в группе не может быть меньше двух переключателей (как можно выбрать что-либо одно из чего-либо одного?). Еще одно следствие заключается в том, что у флажка есть три состояния (выбранное, не выбранное, смешанное), а у переключателя только два, поскольку смешанного состояния у неё быть просто не может (нельзя совместить взаимоисключающие параметры).

В группе переключателей как минимум один должен быть проставлен по умолчанию. Всякий раз, когда пользователю нужно предоставить выбор между несколькими параметрами, можно использовать либо флажки, либо переключатели. Если параметров больше двух, выбор прост: если параметры можно комбинировать, нужно использовать флажки (например, текст может быть одновременно и жирным и курсивным); если же параметры комбинировать нельзя, нужно использовать переключатели

(например, текст может быть выровнен или по левому, или по правому краю).

Если же параметров всего два и при этом параметры невозможно комбинировать (т.е. либо ДА, либо НЕТ), решение более сложно. Дело в том, что группу из двух переключателей часто можно заменить одним флажком.

Предположим, что нужно дать пользователю выбор: показывать в документе линейки или не показывать. В этом случае логично поместить в диалоговое окно рамку группировки со словами «Показывать линейки», а в эту рамку поместить два переключателя: Да и Нет. Понятно, что это решение очень тяжеловесно. Можно сделать проще: убрать рамку группировки и переключатели, а на их место поместить всего один флажок со словами «Показывать линейки». В этом случае все будет хорошо. К сожалению, этот метод работает не всегда. Поскольку в самом флажке написано только то, что произойдет после его включения, но не описано, что произойдет, если его не включить, такая конструкция не работает в ситуациях, когда пользователям по той или иной причине функциональность не поставленного флажка может быть непонятна. Например, если нужно спросить пользователя, в какой кодировке посылать ему письма, не получится заменить два переключателя Windows 1251 и KOI-8 единым флажком KOI-8. Пользователь не обязан понимать, в какой кодировке система будет посылать ему письма по умолчанию. К счастью, такие ситуации редки.

И флажки и переключатели желательно расставлять по вертикали, поскольку это значительно ускоряет поиск нужного элемента.

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

Желательно флажки и переключатели в группе располагать вертикально, поскольку это облегчает поиск конкретного элемента.

Текст подписей. Каждая подпись должна однозначно показывать эффект от выбора соответствующего элемента.

Поскольку флажки и переключатели 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 флажкам и переключателям лучше стараться делать примерно одинаковой длины. Все подписи обязаны быть позитивными (т.е. не содержать отрицания).

Вариант для панелей инструментов. Как флажки, так и переключатели, бывают двух видов: описанные выше стандартные, и предназначенные для размещения на панелях инструментов.

У них есть определенный недостаток: они не различаются внешне (ни в одной ОС метода визуального различения не выработано). Это не очень критично, поскольку панелями инструментов пользуются в основном сравнительно опытные пользователи, так что страдать по этому поводу не стоит. Тем не менее, на панелях инструментов полезно располагать группы переключателей отдельно от групп флажков (чтобы они не смешивались в сознании пользователей).

Графические версии флажков и переключателей 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 флажки и переключатели могут быть кнопками прямого действия.

Списки

Все часто используемые списки функционально являются вариантами флажков и переключателей. Списки бывают пролистываемыми и раскрывающимися, причем пролистываемые могут обеспечивать как единственный (аналогично группе переключателей), так и множественный выбор; раскрывающиеся же работают исключительно как переключатели.

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

В идеале 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, отображая только (...).

Раскрывающиеся (выпадающие) списки (Drop-down List Box). Самым простым вариантом списка является раскрывающийся список, который обладает одним существенным достоинством. Оно заключается в том, что малая высота списка позволяет с большой легкостью визуально отображать команды, собираемые 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.

Подобно списку единичного выбора, выпадающий список предусматривает возможность выбора единственного пункта; различие заключается в том, что выпадающий список отображается на экране только по требованию пользователя. Когда список свернут, в его окне отображается только выбранный пункт. Чтобы изменить выбор пользователь должен открыть список.

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

Первая проблема заключается в том, что иногда отсутствие места на экране не позволяет использовать ни флажки с переключателями, ни пролистываемые списки множественного выбора. Приходится делать раскрывающийся список, в котором помимо собственно элементов есть «метаэлемент», 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 метаэлемент нужно снабжать названием, например, «Все значения» или «Ничего».

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

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

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

Списки единственного выбора (Single Selection List Box). Используется для выбора только одного пункта в списке. Список единственного выбора является промежуточным вариантом между группой радиокнопок и раскрывающимся списком. Он меньше группы радиокнопок с аналогичным числом элементов, но больше раскрывающегося списка. Соответственно, использовать его стоит только в условиях «ленивой экономии» пространства экрана.

Списки расширенного выбора (Extended List Box). Может быть использован для выбора единственного пункта или единственной области, хотя по умолчанию предназначен для выполнения непересекающегося выбора.

Модифицируемый список (List View Control). Представляет собой особую форму расширенного списка, который отображает набор пунктов, каждый из которых содержит пиктограмму и текстовую метку. Содержимое модифицируемого списка может быть представлено в одном из четырех видов: пиктограмма, маленькая пиктограмма, список, отчет (таблица). Пример такого списка – содержимое папки Windows.

Модифицируемое дерево (Tree View Control). Является частным случаем модифицируемого списка, в котором содержимое отображается с учетом логического и иерархического соотношения

между пунктами списка. В таком списке имеются кнопки, которые позволяют изменять форму представления структуры списка в целом и/или отдельных пунктов: они могут отображаться либо в развернутом, либо в свернутом виде. Модифицируемое дерево обычно используется в тех случаях, когда необходимо отобразить отношение между набором контейнеров или других иерархических элементов (пример – справочная система Windows).

Списки множественного выбора (Multiple Selection List Box). Предназначены для независимого выбора нескольких пунктов (подобен прокручиваемому списку флажков). С точки зрения дизайна интерфейсов, списки множественного выбора интересны, прежде всего, тем, что их фактически нет в интернете. Технически создать список множественного выбора непроблематично, для этого в HTML есть даже специальный тег.

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

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.

Комбинированный список, выпадающий комбинированный список – Комбобоксы (Combo Box, Drop-down Combo Box). Комбобоксами называются гибриды списка с полем ввода: пользователь может выбрать существующий элемент, либо ввести свой.

Комбобоксы бывают двух видов: раскрывающиеся и расширенные. Оба типа имеют проблемы.

Раскрывающиеся 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. Кроме того, раскрывающиеся комбобоксы отсутствуют в интернете как класс. Поддержки их нет ни в браузерах, ни в HTML.

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

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

Поля ввода – текстовые поля

Вместе с командными кнопками, флажками и переключателями, поля ввода (Check-box, Rich-text box) являются основой любого интерфейса. В результате требований к ним довольно много.

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

С размерами по горизонтали интереснее. Конечно, ширина поля должна соответствовать объему вводимого текста, поскольку гораздо удобнее вводить текст, который видишь. Менее очевидным является другое соображение: ширина поля ввода не должна быть больше объема вводимого в поле текста, поскольку частично заполненное поле выглядит как минимум неряшливо.

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

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

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

Подписи. Вопрос «где надо размещать подписи к полям ввода?» является одним из самых популярных среди программистов. Один из подходов состоит в том, что, поскольку восприятие подписей занимает определенное время, лучше всего действует следующее простое правило: в часто используемых экранах подписи должны быть сверху от поля (чтобы их было легче не читать), в редко же используемых подписи должны быть слева (чтобы всегда восприниматься и тем самым сокращать количество ошибок).

Подписи к полям ввода имеют определенное отличие от других подписей. В полях ввода подписи можно размещать не рядом с

элементом, а внутри него, что позволяет экономить пространство экрана. Подпись при этом выводится в самом поле ввода, точно так же, как и текст, который в него нужно вводить. Необходимо только отслеживать фокус ввода, чтобы при установке фокуса в поле убирать подпись. Это решение, будучи нестандартным, плохо работает в ПО, но неплохо работает в интернете. Если очень жалко экранное пространство, этим методом стоит пользоваться.

Дискретное текстовое поле (Spin Box – «Крутилка»). Дискретное текстовое поле представляет собой текстовое поле, которое может принимать только ограниченный набор дискретных упорядоченных значений, образующих замкнутую последовательность. Данный элемент интерфейса является комбинацией текстового поля и специального элемента управления, который состоит из двух взаимосвязанных кнопок (он известен также как элемент реверсивного управления – Up-down control). В русскоязычных изданиях этот элемент называют: спин, спиннер, крутилка.

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

Во-первых, чтобы ввести значение в крутилку, пользователю не обязательно бросать мышь и переносить руку на клавиатуру (в отличие от обычного поля ввода). Поскольку перенос руки с место на место занимает сравнительно большое время (в среднем почти половину секунды), 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.

Полосы прокрутки

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

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

Как же сделать пролистывание документа идеальным? Если всё-таки приходится оставлять полосы прокрутки, крайне желательно добиться нескольких свойств полосок:

- Размер ползунка должен показывать общий объем пролистываемого документа.

• Стрелки на полосах должны быть спаренными, т.е. обе стрелки должны находиться рядом, а не на разных сторонах полоски. Это один из случаев, когда логичность интерфейса вступает в противоречие с эффективностью. Если при перелистывании была допущена ошибка, спаренные кнопки позволяют минимизировать перемещение курсора к стрелке, ведущей в обратную сторону.

• Если невозможно сделать динамическое изменение области просмотра при пролистывании, необходимо показывать текущее местоположение пользователя во всплывающей подсказке. Обратите внимание, что местоположение подсказки при перемещении курсора должно оставаться неизменным.

• Необходимо обеспечить обработку погрешности перемещения курсора. Когда пользователь курсором перемещает ползунок, а смотрит в это время на документ, курсор может сойти с полосы. До определённого момента (смещение на 30-70 пикселей) система должна такое смещение игнорировать.

Статус-строка

Строка статуса является, пожалуй, самым недооцененным элементом статуса интерфейса. Распространено мнение, будто строка статуса предназначена для того, чтобы информировать пользователей о значении тех или иных элементов интерфейса. Подразумевается, что если пользователь подводит курсор к какому-либо элементу, в строке статуса появляется краткое его описание. На самом деле строка не может этого делать вообще: дело в том, что курсор находится в одном месте, а подсказка появляется совсем в другом, пользователю при этом приходится читать подсказку либо переводя взгляд, либо периферийным зрением. Разумеется, никто в таких условиях читать подсказку не будет, причем те, кто уверен, что

строка статуса есть место для подсказки, чувствуют это прекрасно. Неудивительно, что разработчики строку статуса игнорируют.

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

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

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

Панель инструментов

Зачастую система обладает функциональностью, которая с одной стороны важна, а с другой – способна свести с ума неподготовленного пользователя. Обычно это касается не столько собственно функций, сколько режимов работы системы. Так, любой не очень опытный пользователь MS Word потратит как минимум минут пятнадцать на выход из режима замены текста, предварительно

нечаянно этот режим включив. Правильнее всего, конечно, включать такие режимы из меню, рассчитывая, что пользователь вряд ли выберет случайно элемент третьего уровня, но если в этот режим надо входить часто, меню спасать перестает.

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

Все панели имеют следующие достоинства:

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

Но есть и отрицательная сторона: панели инструментов занимают много места на экране, поэтому поместить в них всё, что хочется, невозможно. Решить эту проблему можно двояко. Во-первых, можно (и нужно) помещать в панель только наиболее часто используемые команды. Во-вторых, панель можно сделать зависимой от контекста действий пользователя. Оба способа не противоречат друг другу, так что использовать стоит оба.

Панель инструментов нежелательно делать единственным способом вызова функции

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

Дело в том, что когда приходит время совершить выбор, имея в качестве альтернатив визуальные объекты, «человек выбирающий» чаще всего транслирует эти объекты в звуки, а именно в слова (в голове, разумеется).

Затем эти слова помещаются в кратковременную память, в дело включается собственно сознание (предыдущие этапы проходят на бессознательном уровне) и выбирает нужный объект. Применительно к реальной жизни это значит, что пользователь, глядя на панель с пиктограммами, видит скорее не пиктограммы, а слова. Но не всегда.

Случай 1. Опытный пользователь, уже знающий, где на панели находится нужная кнопка, знающий её значение, при этом выбор действия уже произведён при помощи сложившейся ментальной модели. В такой ситуации слова пользователю уже не важны, важно отличие нужной ему кнопки от остальных. Т.е. такому пользователю даже уже все равно, что на пиктограмме изображено, лишь бы она выглядела максимально контрастно (чтобы ускорить её поиск).

Случай 2. Опытный пользователь, обладающий сложившейся ментальной моделью, но не знающий, где конкретно расположена нужная ему кнопка и как она выглядит. Выбор действия уже произведен, осталось только найти нужную кнопку. При этом пиктограмма оказывается ненужной, так как в качестве матрицы пользователь использовать её не может (поскольку не знает, как она выглядит). Более того, поскольку пользователь ищет слово из содержимого своей кратковременной памяти, каждая пиктограмма будет его без пользы отвлекать, при этом пользователь будет тратить время на расшифровку смысла всех попадающихся ему на пути пиктограмм.

Случай 3. Неопытный пользователь без сложившейся ментальной модели. Такой пользователь большую часть всего времени тратит на поиск нужной ему кнопки, а также, поскольку каждая кнопка ему внове, на постоянное улучшение своей ментальной модели системы с учетом своих новых открытий. В таких случаях пиктограммы лучше текста, но не заменяют его, так как помогают быстрее понять действие кнопки (в том, разумеется, случае, когда пиктограмма адекватна смыслу действия).

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

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

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

Индикатор состояния процесса и область сообщений

Индикатор состояния процесса (Progress Indicator) обычно используется для того, чтобы отобразить ход выполнения какой-либо длительной операции (процесса). Он состоит из прямоугольной зоны, которая «заполняется» слева направо.

Индикатор не является интерактивным элементом, он только отображает информацию. Полезно снабдить его текстом, поясняющим его назначение (текст располагается вне индикатора).

Индикатор используется в качестве средства обратной связи для длинных операций или фоновых процессов.

В области сообщений (System Tray) приложение может поместить специальный индикатор или сообщение, уведомляющее пользователя о какой-либо ситуации; выведенное сообщение сохраняется в области сообщений даже тогда, когда приложение находится в неактивном состоянии.

Поскольку панель задач – коллективный ресурс, используемый всеми активными приложениями, в область сообщений следует выводить только такую информацию, которая носит «глобальный» характер или необходима пользователям при работе с другими приложениями.

Для отображения информации в области сообщений следует использовать пиктограммы.

При отображении информации в области сообщений придерживаются следующих рекомендаций.

- Обеспечьте появление на экране всплывающего окна, которое содержит дополнительную информацию или средства управления для объекта, представленного индикатором в области сообщений; для вызова окна обычно используют щелчок левой клавишей мыши на изображении индикатора. Если нет необходимости в выводе дополнительной информации, всплывающее окно создавать не стоит.

• Для объекта, представленного индикатором, должно поддерживаться всплывающее меню. Это меню должно содержать основные команды панели свойств соответствующего объекта.

• Создайте всплывающую подсказку для индикатора.

• Предоставьте пользователю возможность не отображать индикатор в области сообщений.

ЮЗАБИЛИТИ-ТЕСТИРОВАНИЕ

Виды юзабилити-тестирования

Тестирование на удобство применения проводится для того, чтобы оценить качество работы продукта и выяснить, насколько он эффективен, рентабелен и довольны ли им пользователи. Такое тестирование является неотъемлемой частью процесса разработки и проектирования продукта.

Юзабилити-тестирование и его оценка должны рассматриваться в бюджете как часть рабочих расходов. На проведение тестирования должно выделяться от 5 до 10% от общего бюджета. Как и другие расходы, в дальнейшем они окупятся повышением доходов, связанных с улучшением качества продукта.

Существуют следующие способы проведения тестирования:

- наблюдение;
- проведение опросов и исследований;
- контекстуальные опросы;
- эвристические оценки;
- работа с выделенными группами;
- лабораторное тестирование.

Важной частью любого тестирования является правильный подбор пользователей, и их достаточного количества. Участники теста должны быть типичными представителями пользователей

данного продукта. Их количество зависит от различных факторов: время, ресурсы, проект тест, тип тестируемых задач и вид статистического анализа результатов.

Для обнаружения основных проблем удобства применения достаточно задействовать 4-8 участников. Если по завершении тестирования возникают новые проблемы, то количество участников следует увеличить.

Методы оценки удобства применения необходимо выбирать в зависимости от целей и задач, стоящих перед продуктом. Как правило, юзабилити-тестирование включает два типа методов оценки.

- Методы оценки работы, подразумевающие подсчет действий, определение полноты выполнения задачи, подсчет времени, затраченного на это выполнение, ошибок и обращений за помощью. Такие методы называют *численными*.
- Субъективные методы, включающие сбор устных и письменных сообщений пользователей об их восприятии, мнениях, суждениях, предпочтениях, а также степени удовлетворенности от системы и их собственной выполненной работы. Эти методы носят название *качественных*.

Цели и задачи

Прежде чем планировать и проводить тестирование на удобство применения продукта, следует четко определить цели и задачи, стоящие перед ним. Эта задача лежит на владельцах продукта, специалистах, составляющих планы, проектировщиках и разработчиках.

Перечислим факторы, которые влияют на удобство применения:

1) *Полезность* – степень, до которой продукт позволяет пользователю достичь стоящей перед ним цели (могут ли

пользователи использовать продукт?). Оценка полезности осуществляется с помощью оценки качества выполненной работы.

2) *Эффективность* – насколько успешно продукту удается содействовать пользователю в выполнении стоящей перед ним задачи (насколько хорошо пользователь может выполнить стоящую перед ним задачу с помощью данного продукта?). Оценка эффективности осуществляется с помощью оценки качества выполненной работы.
3) *Простота изучения*. Пользователи могут начать использовать продукт, приобретя определенный уровень знаний после прохождения тренинга (насколько хорошо обучены пользователи?). Оценка простоты изучения осуществляется с помощью оценки качества выполненной работы.
4) *Отношение пользователей* – восприятия, ощущения и мнения пользователей по поводу изучения и использования данного продукта (что пользователи думают об удобстве применения продукта?). Оценка осуществляется по отзывам пользователей (устной или письменной обратной связи).

Цели и задачи, стоящие перед разработкой удобства применения, должны определяться для всех программных продуктов. *Цели* – это обеспечение преимуществ перед конкурирующими продуктами в области простоты изучения, эффективности, гибкости и т.д. Таким образом, цели напрямую связаны с вышеперечисленными факторами.

Цели сами по себе не подлежат непосредственной оценке – они должны делиться на задачи. *Задачи* – это уточнение целей, они более конкретны и детальны, их можно оценить и измерить. Достижение одной цели может потребовать решения множества задач. Задачи

должны быть выстроены таким образом, чтобы содержать информацию по конкретным действиям или операциям.

Пример характеристики задач.

Задача удобства применения	**Критерий**	**Качество работы**	**Условия**
После 4-часового тренинга 90% пользователей в состоянии выполнить заказ клиента в течение 5 мин.	90% пользователей; в течение 5 мин.	Выполнить заказ клиента	После 4-часового тренинга
Полезность: После выполнения пяти сценариев задач 90% пользователей будут в состоянии успешно выполнить задачу	90% пользователей; выполнить задачу	Успешно выполнить задачу	После выполнения пяти сценариев задач
Эффективность: После выполнения пяти сценариев задач 75% пользователей будут в состоянии успешно выполнить задачу в течение 10 мин.	75% пользователей; в течение 10 мин.	Успешно выполнить задачу	После выполнения пяти сценариев задач
Простота изучения: После 4-часового тренинга все пользователи достигнут определенного уровня владения продуктом	Все пользователи; определенный уровень владения продуктом	Успешно овладеть продуктом	После 4-часового тренинга
Отношение пользователей: После выполнения пяти сценариев задач 85% пользователей оценят степень своей удовлетворенности продуктом на 5,5 и выше	85% пользователей; степень удовлетворенности 5,5 баллов по 7-балльной системе	Степень удовлетворенности	После выполнения пяти сценариев задач

баллов (по 7-балльной системе)			

Надежность и достоверность результатов

Надежность теста состоит в том, что один и тот же тест может проводиться снова и снова и будет демонстрировать одинаковые результаты. Надежность отличается от достоверности теста. Тест считается достоверным, если точно измерено то, что должно быть измерено. Сравнительное тестирование на удобство применения продуктов может быть малонадежным. Несложно разработать сценарии и задачи, которые выгодно выставляют технологию или функции одного продукта по сравнению с аналогичными характеристиками другого. Различные типы измерений, используемых при тестировании, могут радикально изменить результаты.

Приведем рекомендации по тестированию:

- сравнительные тесты могут представить полезную информацию при исследовании удобства применения новых версий программных продуктов;
- используйте задачи общего характера и области известных проблем в качестве базовых задач, чтобы произвести точные сравнительные измерения между версиями продуктов;
- применяйте стандартные измерения удобства применения, например, качество выполнения работ пользователем (выполняемость задач и показатель успешности их выполнения), степень удовлетворенности пользователей (данные по степени удовлетворенности и предпочтений);
- используйте внешних, независимых производителей для планирования и проведения тестирования на удобство применения, чтобы уменьшить субъективность восприятия.

Тесты на удобство применения должны разрабатываться, а их результаты анализироваться для того, чтобы совершенствовать интерфейсы продуктов и повышать производительность пользователей, а не для сравнения и сопоставления с другими продуктами и интерфейсами.

Отчетная карточка теста

В отчетной карточке перечисляют все основные темы, которые должны рассматриваться в любом тесте на удобство применения. Карточка используется при разработке собственного теста, а также для правильной оценки исследований на удобство применения.

Темы теста на удобство применения	**Возможные вопросы**
Спонсор теста/ Исполнитель теста	Кто спонсирует тест на удобство применения? Кто действительно проводит тест?
Цели и задачи теста	Описываются ли задачи теста? Описываются ли цели теста? Было ли дано операционное определение задачам (можно ли их измерить)?
Проект теста и сами процедуры	Пригоден ли проект теста, учитывая заданные цели и задачи? Следует ли тест действующей методологии разработки экспериментов? Каковы процедуры теста (введение, тренинг, задачи, анкеты, опросы и т.д.)?
Программная и аппаратная платформа	Действительно ли тестируются требуемые продукты? Действительно ли используется соответствующая аппаратная платформа? Действительно ли компьютеры сконфигурированы аналогичным образом (скорость процессора, память, хранение и т.д.)?

Участники теста	Кто участвует в тесте? Каковы их демографические данные (возраст, пол и т.д.). Каков уровень их навыков в работе с компьютером? Какие приложения и операционные системы они использовали? Где и каким образом были выбраны участники теста? Какие методы и критерии использовались для подразделения участников по категориям (начинающие, опытные и т.д.)? Как участники теста были разделены по группам? Оплачивали ли участникам теста их участие?
Задачи теста	Каковы задачи теста? Являются ли сценарии задач неясными или наоборот, подсказывающими пользователям их действия? Используются ли в тесте актуальные данные или данные тестов? Соответствуют ли задачи целям теста? Выполняются ли задачи всеми участниками теста? Выполняются ли задачи на всех компьютерах? Не сориентированы ли задачи под определенный продукт?
Поддержка и помощь в тестировании	Прошли ли участники теста предварительный тренинг? Какой? Оказывают ли участникам помощь во время теста? Кто ее оказывает? Можно ли располагать технической поддержкой (аппаратной и программной) во время теста?
Оценки, проводимые во время теста	Что оценивается во время теста (качество работы, предпочтения, наблюдения)? Соответствуют ли оценки, полученные во время тестирования, целям и задачам теста? Каким образом собираются оценки (самооценки, хронометраж, видеозаписи и т.д.)?
Критерии теста	Каковы критерии каждой оценки? Что определяет успешность выполнения задач?

	Что определяет помощь? Что определяет ошибку пользователя?
Анализы теста, результаты и выводы	Доступны ли оригинальные данные теста? Определен и описан ли анализ теста? Пригоден и достоверен ли статистический анализ по результатам теста? Существенны ли статистически результаты теста? Основаны ли выводы по тесту на его результатах? Можно ли сделать общие выводы из проекта теста и его результатов? Соответствуют ли выводы теста целям и задачам теста?
Надежность против достоверности	Надежен ли тест на удобство применения (можно ли повторить результаты)? Достоверен ли тест на удобство применения (действительно ли тест и результаты дают оценку тому, что должно быть оценено)? Какова реакция со стороны на проект теста и его результаты (в отношении надежности и достоверности)?
Этическая сторона проведения теста	Сообщили ли участникам теста о задачах, оценках и самой процедуре проведения теста (формы согласия, видеозаписи, наблюдатели и т.д.)? Разрешено ли пользователям по желанию прекратить участие в тесте? Подталкивают ли пользователей к использованию определенных функций или выполнению определенных задач? Советуют ли пользователям не торопиться или выполнять задачи как можно быстрее?

Чем больше пользователей зависят от неудобного в применении проекта, тем сложнее его изменить. Необходимо учесть, что людям нравится то, что им знакомо. Пользователи могут адаптироваться к плохо разработанным проектам, хотя они не должны этого делать.

Анкета по словам

Ниже приведен пример анкеты по словам, которую должен выбрать респондент после прохождения тестирования для описания опробованного продукта. Слова в анкете специально записаны вразнобой, именно так и нужно предъявлять их респондентам.

Неудобный – Устаревший – Эффективный – Замусоренный – Яркий – Тусклый – Привлекательный – Чистый – Прямой – Ясный – Непоследовательный – Хороший – Холодный – Неуправляемый – Стандартный – Управляемый – Интуитивный – Веселый – Умный – Любительский – Неэффективный – Опасный – Безопасный – Скучный – Радостный – Жесткий – Раздражающий – Треугольный – Комфортабельный – Бесполезный – Халтурный – Качественный – Теплый – Светлый – Последовательный – Гибкий – Загадочный – Интересный – Ненадежный – Красивый – Глупый – Некрасивый – Непривлекательный – Полезный – Запутанный – Современный – Удобный – Понятный – Непредсказуемый – Тяжелый – Легкий – Дружественный – Нестандартный – Плохой – Профессиональный – Надежный – Сложный – Простой – Темный – Недружественный – Медленный – Круглый – Печальный – Быстрый – Предсказуемый – Непонятный – Головоломный – Грустный

Формальная анкета

Анкета представляет собой несколько вопросов, для каждого из которых респондент может выбрать один из пяти вариантов ответа. Анкета спроектирована только как пост-тестовая, ее применение в другом качестве сомнительно.

Вопросы анкеты:

Во время выполнения заданий я редко ошибался	Нет	○	○	○	○	○	Да
Система способна делать все, что мне нужно и даже больше	Нет	○	○	○	○	○	Да
Система работает достаточно быстро	Нет	○	○	○	○	○	Да
Мне нравится внешний вид интерфейса	Нет	○	○	○	○	○	Да
Я чувствую, что если я лучше изучу систему, я смогу делать в ней вещи, о которых сейчас даже и не подозреваю	Нет	○	○	○	○	○	Да
Систему можно легко настроить под мои нужды	Нет	○	○	○	○	○	Да
Начать работу было легко; я не столкнулся с существенными трудностями	Нет	○	○	○	○	○	Да
Всякий раз, когда я ошибался, я с легкостью замечал и исправлял свою ошибку	Нет	○	○	○	○	○	Да
Я доволен своей скоростью работы	Нет	○	○	○	○	○	Да
Во время выполнения заданий я чувствовал себя вполне уверенно	Нет	○	○	○	○	○	Да
В любой момент времени я понимал, что я должен сделать дальше	Нет	○	○	○	○	○	Да

Результаты нужно подсчитывать по следующему алгоритму: центральное значение дает ноль баллов, крайние значения дают либо –2 балла (левый вариант ответа), либо +2 балла (правый вариант), промежуточные значения либо –1 либо +1 балл соответственно. Сумма баллов является сравниваемым значением.

Тестовые задания и требования к ним

Тестовые задания, помимо того, что должны соответствовать пользовательским задачам, должны обладать еще и следующими свойствами.

Однозначность. Задания должны быть сформулированы так, чтобы исключить их неправильное толкование респондентом. Если респондент поймет задание неправильно, почти наверняка не удастся по ходу теста направить его на правильный путь, не подсказав ему одновременно последовательности выполнения задания.

Полнота. В тексте задания должна присутствовать вся информация, необходимая для выполнения этого задания.

Краткость. Если замеряется скорость выполнения заданий, задания должны быть достаточно краткими, чтобы длительность чтения заданий респондентами не влияла на продолжительность выполнения самих заданий (люди читают с разной скоростью). Если текст задания будет велик по объему, придется вручную отсекать длительность чтения для каждого задания, что очень трудоемко.

Отсутствие подсказок. По тексту задания не должно быть понятно, как это задание нужно выполнять. Например, недопустимо использовать терминологию системы – вместо каждого термина нужно расписывать его значение, иначе респонденты просто нажмут кнопки с теми же словами и никаких проблем не будет выявлено. В задании должны присутствовать точка начала выполнения задания, т.е. должно быть прописано то окно или экран, на котором респондент должен находиться в начале. Если такой информации представлено не будет, респонденты неизбежно будут переходить к другим фрагментам интерфейса, а значит, задание разными респондентами будет выполняться по-разному, что делает бессмысленным все статистические расчеты. Фиксировать начальную точку задания нужно еще в конце предыдущего задания. Если задание начинается с чистого листа, в конце предыдущего задания должно

быть написано «вернитесь на главный экран». Если задание должно начинаться с места, на котором закончилось предыдущее задание, предыдущее задание должно заканчиваться словами «закончив, не закрывайте текущее окно/останьтесь на этом экране».

Помимо этих общих требований нужно учитывать еще и следующее.

- Возможно, что на одну пользовательскую задачу нужно будет написать несколько тестовых заданий. Типичный случай – задача слишком велика, чтобы ее можно было вместить в одно задание. Кроме того, если пользовательская задача является частотной, не очень важно, как она выполняется в первый раз – гораздо интереснее узнать, как пользователи будут выполнять ее во второй, третий и т.д. разы. В этом случае в пределах теста на одном респонденте прогонять эту задачу потребуется несколько раз, каждый раз варьируя задания.
- Помимо заданий, в которых респондент должен выполнить какое-либо действие, допустимы и желательны двойные задания, в которых респондент должен сначала решить, нужно ли ему в данное время это действие выполнять. Например, если тестируется программа дефрагментации диска, вместо задания «Дефрагментируйте диск компьютера» лучше использовать задание «Проверьте степень фрагментации диска и, если вы сочтете это необходимым, дефрагментируйте диск компьютера». Такие задания должны быть составлены таким образом, чтобы респондент не мог отказаться от принятия решения, не глядя сказав, что все хорошо и дефрагментация не нужна. Кроме того, перед таким тестом разумно намеренно фрагментировать диск, чтобы респондент не смог избежать задания.

- Порой по ходу задания нужно специально изменить состояние системы. К примеру, если нужно узнать, как именно пользователи решают определенную проблему, придется эту проблему создать. Прерывать для этого выполнение теста недопустимо, поскольку это отвлечет респондента. В таких случаях перед соответствующим заданием можно вставить другое задание, в котором респондент должен создать проблему самостоятельно. Разумеется, никаких сведений об интерфейсе такое задание не предоставит.
- Анализ результатов и подведение статистики сильно упрощаются, если делать не малое число длительных заданий, а большое число заданий кратких, требующих перемещения всего на пару экранов или заполнения одной–двух форм.
- Первое задание теста должно быть вводным, предназначенным исключительно для введения респондента в процесс. Соответственно, оно должно быть простым, а его результаты можно не учитывать.

Не забудьте проверить, что ваши сценарии могут быть выполнены респондентами за ожидаемое время теста. Вероятно, список сценариев придется сокращать.

Подготовка отчета

В общем виде оптимальной структурой отчета является:

1) Резюме;
2) Основные проблемы (интерфейсные проблемы, проявляющиеся по всему интерфейсу);

3) Частные проблемы (проблемы, проявляющиеся на отдельных экранах);
4) Количественные данные (если они собирались);
5) Приложение 1. Методика эксперимента и условия теста;
6) Приложение 2. Описание тестовых сценариев;
7) Приложение 3. Описание респондентов.

СТАНДАРТИЗАЦИЯ

Стандартизация пользовательского интерфейса

Уже в середине 70-х годов специалисты в области человеко-машинных компьютерных систем осознали необходимость формирования единых подходов к реализации пользовательского интерфейса. В настоящее время существуют специальные организации, отвечающие за разработку стандартов.

Реальное влияние на процесс стандартизации технологий, используемых в интернете, оказывают следующие организации и группы.

1. *World Wide Web Consortium (W3C)* – создан в 1994 г., представляет интересы пользователей, научных учреждений и компьютерных фирм и имеет солидную международную репутацию. W3C предлагает и поддерживает веб-технологии, публикует коды эталонных реализаций. Работает с технологиями браузеров HTML и XML, и рядом стандартов (HTTP, URL-адреса, цифровые сертификаты).
2. *International Organization for Standartization (ISO)* – Международная организация по стандартизации, которая занимается разработкой стандартов по большому спектру продуктов и технологий в разных странах.

Поскольку разработка интерфейса является необходимой частью разработки любого ПО, основные требования к интерфейсам можно найти в стандартах, посвященных жизненному циклу программных средств. Примерами таких стандартов являются следующие:

- MIL-STD-498 – стандарт разработки ПО Министерства обороны США;
- ISO/IEC 12207:1995 («Процессы жизненного цикла программных средств»);
- В России – комплект документов «Единая система программной документации» (появился в 1977 г. и постоянно корректируется).

На основе существующих стандартов разрабатываются руководящие принципы, инструкции и нормативы по разработке. Принципы содержат основополагающие требования. Инструкции относятся к элементам представления информации и взаимодействия и представляют собой правила и объяснения для создания элементов интерфейса и внешнего вида.

Руководящие принципы, отраженные в инструкциях, должны позволять пользователю применять к интерфейсу свое знание реального мира. Интерфейс должен иметь схожее поведение с объектами реального мира.

Инструкции и нормативы распределяются в порядке их важности по отношению к пользовательскому интерфейсу. Каждое руководство содержит разделы «когда используется» и «как используется». Проектировщики должны следовать всем инструкциям для обеспечения базисного уровня целостности интерфейса.

Решения о дополнительных составляющих нормативов должны приниматься на основе индивидуального или корпоративного стиля, распределения задач, ресурсов, бюджета.

Стандарты и руководящие принципы являются строительными блоками, на которые должны опираться все ваши усилия по разработке и проектированию. Однако строгое следование им не обязательно приведет к созданию удобного интерфейса. Разработка интерфейса – больше искусство, чем наука.

Необходимость стандартов

Использование стандартов предоставляет следующие выгоды.

Эффективность кода. Повторяемый код можно разделить на контент HTML и информацию о стиле оформления (CSS) и поведении (JavaScript), что позволит уменьшить размеры файлов, а написанный однажды код использовать повторно, когда это понадобится.

Легкость сопровождения. Можно написать код HTML один раз, а затем применять стили оформления и поведение когда они понадобятся с помощью классов и функций. В случае необходимости изменить что-то в будущем, можно сделать изменение в одном месте, и оно распространится на весь сайт, вместо того, чтобы определять изменение везде, где оно потребуется!

Доступность. Одной из значительных проблем является создание сайтов доступными для всех, кто-бы это ни был, невзирая на обстоятельства. Это включает создание сайтов, доступных для людей с физическими недостатками, такими как слепота, ослабленное зрение и двигательная ограниченность (т.е. людей, которые ограничены в движениях, не могут полноценно использовать свои глаза и руки, или вообще не могут их использовать). Используя стандарты и эффективные методы, можно без дополнительных усилий сделать сайты доступными для этой значительной группы пользователей.

Совместимость с устройствами. Под этим понимается обеспечение того, что сайты будут работать не только на различных платформах (т.е. Windows, Mac, Linux), но также на альтернативных устройствах просмотра, которые сегодня могут включать мобильные телефоны, телевизоры и игровые консоли. Эти устройства имеют некоторые ограничения, такие как размер экрана, вычислительная мощность, доступные механизмы управления и многие другие, но и в этом случае, используя стандарты и эффективные методы, можно в значительной степени гарантировать, что сайты будут работать на большинстве этих устройств.

Web-роботы/поисковые системы. Создание сайтов как можно более заметных для так называемых роботов, которые просматривают сеть и индексируют сайты, что позволяет им улучшать свое положение в результатах на поисковых сайтах.

Часто люди считают, что разработка на основе стандартов требует слишком много времени для изучения использования стандартов Web вместо устаревших методов, и разработки сайтов, которые работают во всех браузерах. Почему же не начать с изучения правильного способа сделать это, и избежать различных проблем в будущем? Если вы решили научиться создавать сайты, то почему бы не научиться делать это правильно?

Рябченко Алексей Иванович
Лукьяненко Владимир Олегович

ИНФОРМАЦИОННАЯ АРХИТЕКТУРА И ЮЗАБИЛИТИ

**Курс лекций
по одноименной дисциплине для слушателей
специальности 1-40 01 74 «Web-дизайн
и компьютерная графика»
заочной формы обучения**

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 10.12.12.
Рег. № 49Е.
http://www.gstu.by