南京大学
NANJING UNIVERSITY

人机交互的软件工程方法
—— 可视化设计

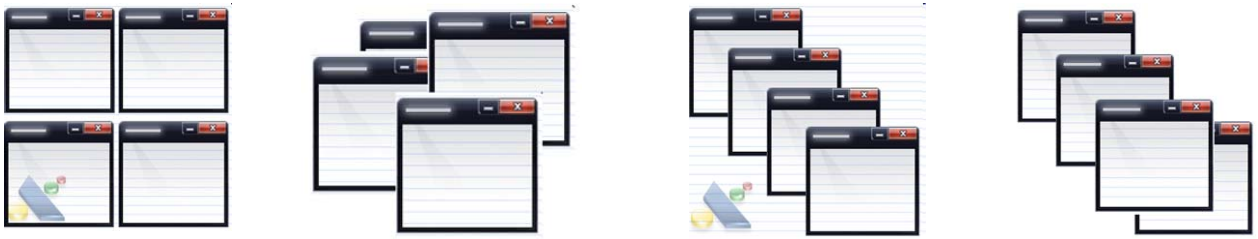
主讲教师：冯桂焕
fgh@software.nju.edu.cn
2012年春季

主要内容

- 窗口和菜单
- 对话框
- 常用控件
- 工具栏
- 屏幕复杂性度量
- 用户界面设计原理

窗口Window

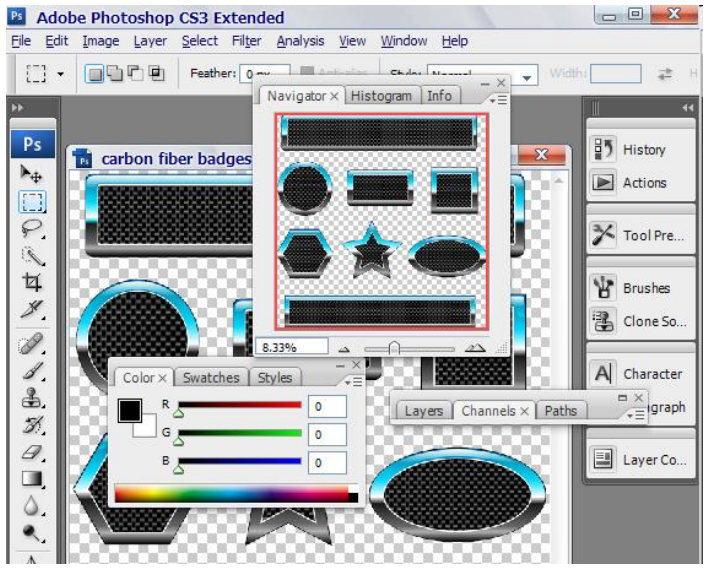
- 源于Xerox Alto系统，后被融入Apple操作系统和Windows操作系统
- 状态
  - 最大化
  - 最小化
  - 还原



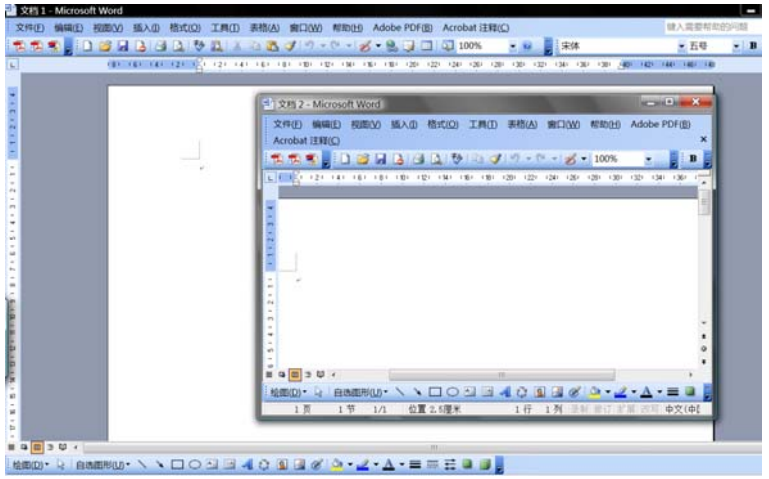
    - 平铺(Tile)窗口：允许拖放操作
    - 重叠(Overlapping)窗口：有效利用屏幕空间
    - 层叠(Cascade)窗口：可视化组织各窗口

窗口界面类型

- 多文档界面
  - 优点
    - 节省系统资源
    - 最小的可视集
    - 协同工作区
    - 多文档同时可视化
  - 缺点
    - 菜单随活动文档窗口状态变化，导致不一致性
    - 文档窗口必须在主窗口内部，减弱多文档显示优势
    - 屏幕显示复杂：子窗口可能在父窗口中被最小化



- 单文档界面
  - 优点
    - 从用户角度出发，以文档为中心
    - 界面的视觉复杂性小
  - 缺点
    - 不能管理分散但相关的文档窗口
    - 相关文档不能从相同类型的其他文档中分离
    - 文档打开过多时，任务栏可能被占满

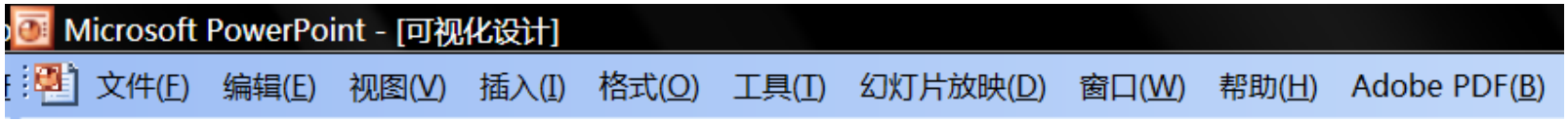


- 标签文档界面
  - 窗口菜单：包含了当前打开窗口的列表
- 优点：让用户看到哪些窗口是打开的
- 缺点：不允许用户看到两个及以上的窗口内容

菜单

- 访问系统功能的工具，已经成为窗口环境的标准特征
  - 某些功能可以立即执行，或由选择的菜单命令激活一个包含相关功能的对话框
  - 菜单适合初学者，包含完整工具集合
    - 不仅仅适用于初学者
- 必不可少的组成部分
  - 菜单标题
  - 菜单选项
  - 最重要的特性：描述性、一致性

- 菜单栏
  - 所有窗口必备的基本组件
  - 菜单栏是代表下拉式菜单的菜单
  - 菜单选项的标签、位置、归类等均已标准化
    - 从左向右：文件、编辑、视图和窗口，帮助位于窗口的最右边
    - 可选菜单常出现在编辑和窗口菜单之间



注意事项

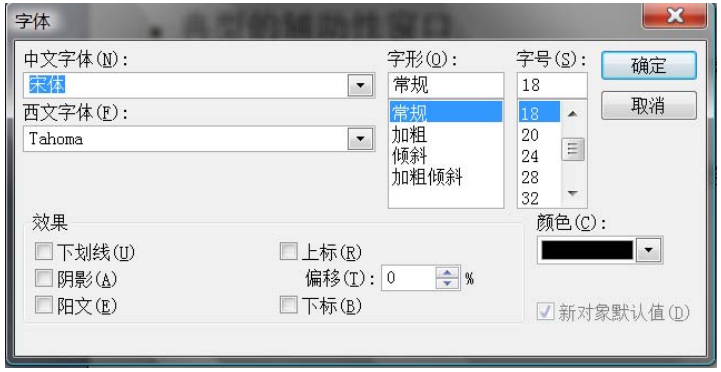
- 菜单应该按语义及任务结构来组织
  - 糟糕的例子：File菜单
- 合理组织菜单接口的结构与层次
  - 菜单太多或太少都表明菜单结构有问题
- 菜单及菜单项的名字应符合日常命名习惯
- 菜单选项列表即可以是有序的也可以是无序的
  - 频繁使用的菜单项应当置于顶部
- 为菜单项提供多种的选择途径
  - 尽可能使用工业标准
- 对菜单选择和点取设定反馈标记
  - 如灰色屏蔽，为选中的菜单项加边框在菜单项前面加√符号等

主要内容

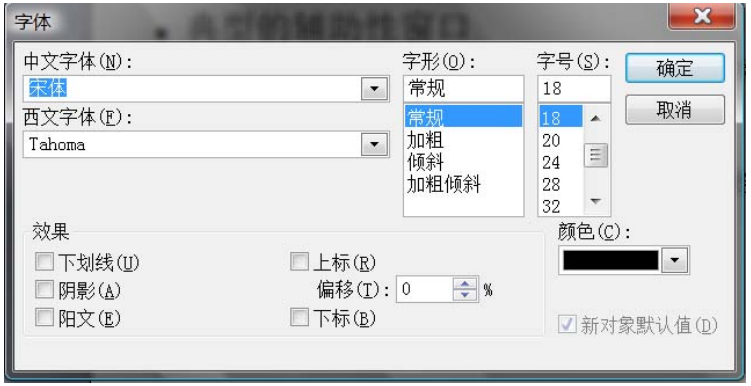
- 窗口和菜单
- 对话框
- 常用控件
- 工具栏
- 屏幕复杂性度量
- 用户界面设计原理

对话框

- 典型的辅助性窗口
  - 常用于将某些破坏性的、令人混淆的、不太常用的操作与主要工作中使用的工具分开考虑
  - 通知用户系统的一个错误或潜在的问题
  - 没有标题栏图标、状态栏和调整窗口大小的按钮
- 分类
  - 模态对话框 vs. 非模态对话框



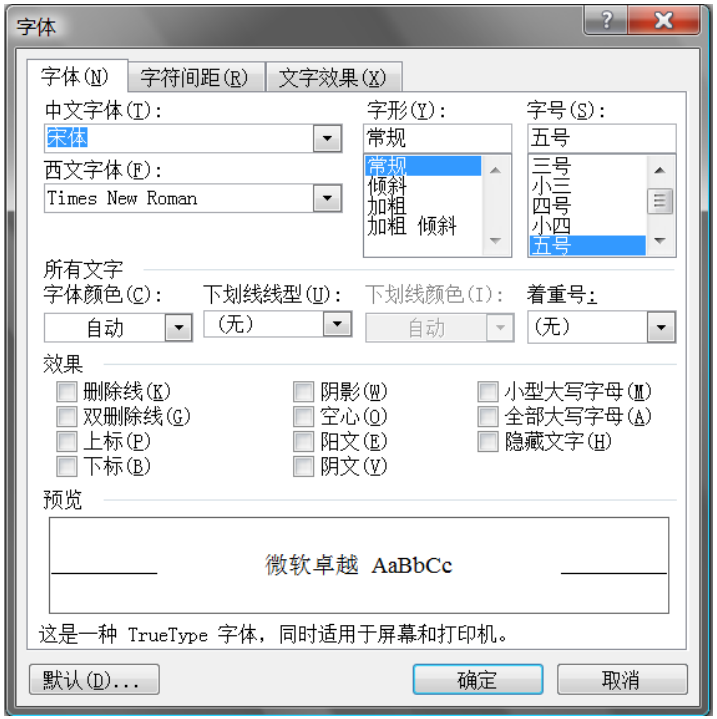
- 模态对话框
  - 冻结了它属于的应用，禁止用户做其他操作，直到处理了对话框中出现的问题
  - 可以切换到其他程序进行操作
  - 用户最容易理解，操作非常清晰
- “应用模态”
  - 只停止其所属的应用程序
- “系统模态”
  - 使系统中的所有程序都停止
  - 大多数情况下，应用程序不应该有系统模态对话框



- 非模态对话框
  - 打开后无须停止进度，应用程序也不会冻结
  - 由于其操作范围不确定而难以使用和理解
  - 举例
    - Word的查找和替换对话框
    - 画图程序
      - 可以在主窗口和非模态对话框之间拖动对象
  - 存在的问题
    - 缺乏一致的终止命令，如取消、应用、关闭等

用途

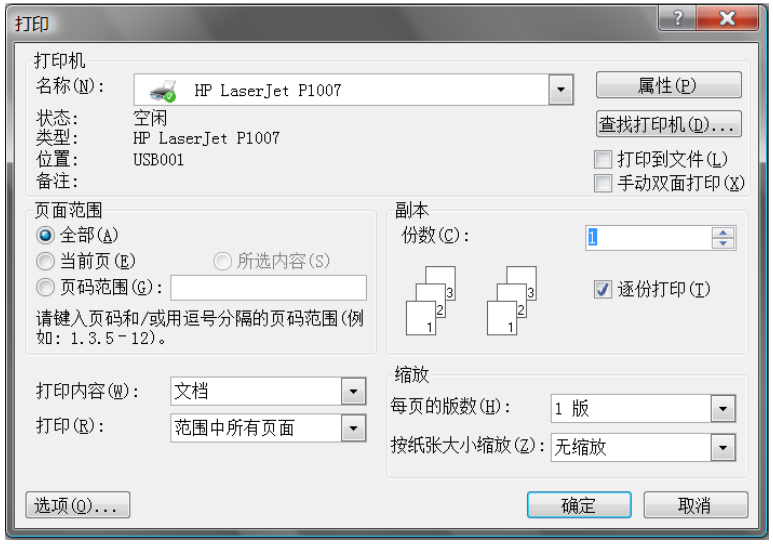
- 属性对话框
  - 呈现所选对象的属性或者设置，并允许用户改变
  - 模态、非模态均可
  - 控制选择遵循“对象-动词”形式
    - 选择对象，通过属性对话框为所选对象选择新的设置



功能对话框

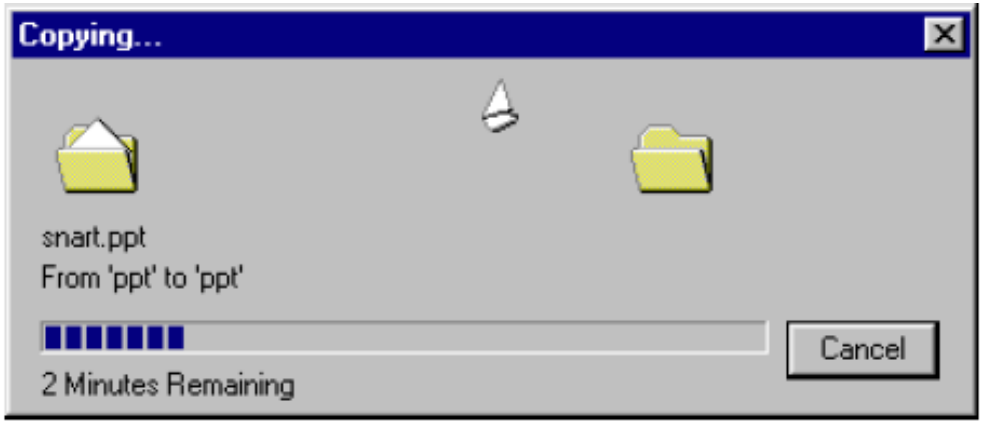
- 控制单个功能，如打印、拼写检查等
- 允许用户开始一个动作，并允许设置动作的细节
  - 打印多少页、多少份、哪一台打印机

- 注意
  - 一个功能可以配置，不意味着用户每次调用都想配置
  - 配置和实际功能最好隔离开



进度对话框

- 由程序启动，而不是根据用户请求启动
  - 向用户清楚地表明正在运行一个耗时的进程
  - 向用户清楚地表明一切正常
  - 向用户清楚地表明进程还需多长时间
  - 向用户提供一种取消操作和恢复程序控制的方式
- 进度表应表明相对整个过程所耗费时间的进度，而不是相对整个过程规模的进度



- 对话框是一个单独的房间
- Mozilla Firefox和微软IE的网页浏览器
  - 加载网页是内在操作
  - 进度条被安放在其下方的状态栏中

公告对话框

- GUI中滥用最多的元素
- 无须请求，由程序直接启动
- 阻塞型公告 VS. 临时公告
  - 错误、警告和确认消息都是阻塞型公告对话框
    - 为程序服务，牺牲用户利益，应尽量避免
  - 绝不要用临时对话框作为错误信息框或确认信息框！

- 特殊的对话框：错误、警告和确认
  - 为什么要消除以及如何消除

错误对话框

- 用户希望避免错误造成的结果，而不是错误消息
- 错误消息存在的问题
  - “用户经常因为产品设计中的错误而自责”—Donald Norman
  - 设计者和程序员角度
    - 当人犯错时喜欢别人告诉他们
    - 错误消息框在提醒用户一些严重的问题
  - 用户角度
    - 愚蠢地停止了程序进度
    - 责备自己之前，先责备发出消息的一方
  - 错误消息不起作用！

消除错误消息

- 用更健壮的软件取代错误消息
  - 消除用户犯错误的可能性
  - 为所有的数据输入使用有界控件
- 为用户提供正面反馈
  - 软件的负面反馈对用户而言是一种侮辱
- 最后一招：改进错误消息框
  - 保证始终有礼貌、具有启发性、还要助人为乐
  - 澄清问题的范围
  - 可选择的方法
  - 默认情况程序会做什么
  - 丢失了哪些信息

警告对话框

- 通知用户程序的行为，“确认”操作赋予用户忽略该行为的权利
  - 停止了进度
- 举例
  - 为什么不把提示信息放在主查找对话框中？
  - 出现一个注定要关闭的对话框有必要吗？
- 建议：软件不必阿谀奉承

确认对话框

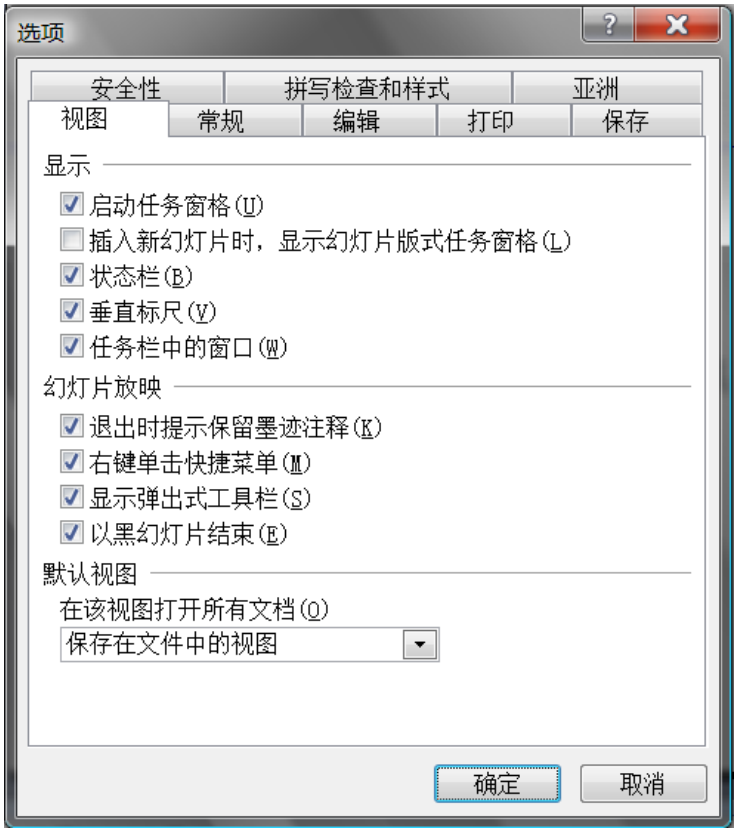
- 程序对自己的行为不自信，用对话框来征求许可
- 总是来自程序，而不是用户
  - 把责任推卸给用户
- 对话框在喊“狼来了”
  - 常规地方提供确认对话框，可使用户忽略真正的意外

消除确认对话框

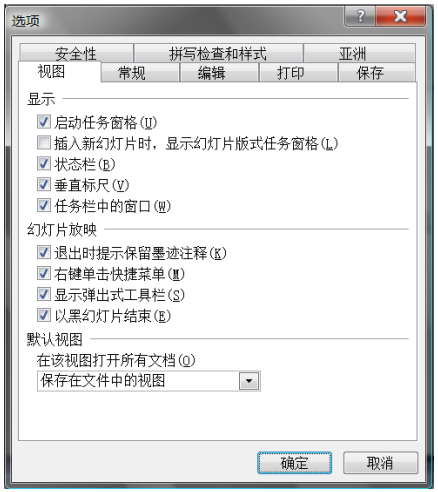
- 原则1：做，不要问
  - 设计软件时，勇往直前给它确信的力量
- 原则2：让所有操作都可以撤销
  - 删除或覆盖文件时可将文件移动到暂时目录，待一段时间或物理删除后再删除
- 原则3：提供非模态反馈帮助用户避免犯错误
  - 如文档页面应显示真实可打印区域的向导，以避免打印文档超出可用的打印区域

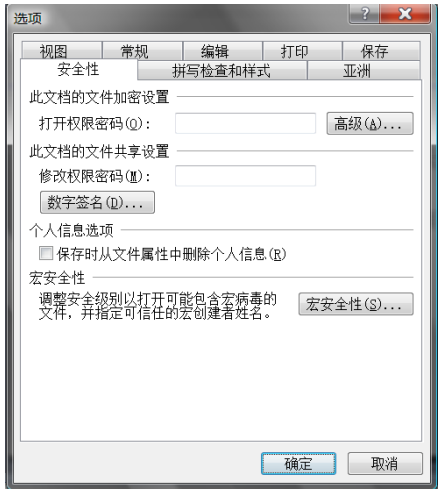
管理对话框内容

- 标签对话框
  - 取代充斥大量控件的大对话框
  - 拥有更多控件不意味着用户会觉得界面易于使用或功能强大
    - 不同窗格内容必须有放在一起的道理
    - 窗格组织为某个专题上的深度或广度增加

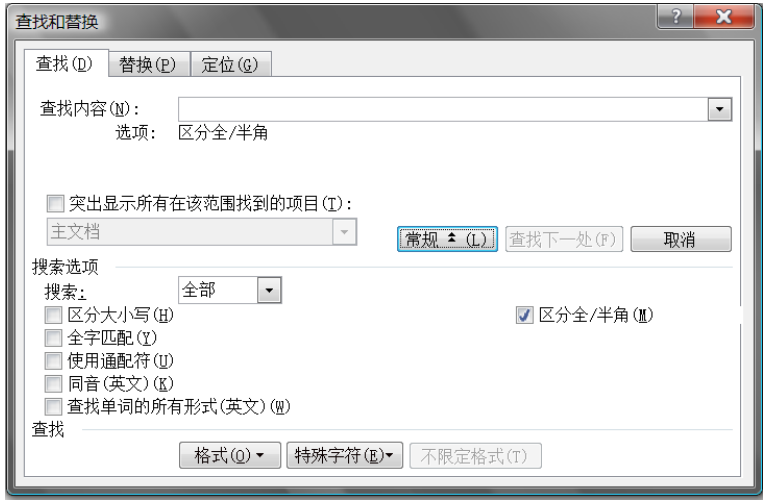


- 标签对话框
  - 成功的原因是其遵循了用户有关“事物存储”的心理模型，即单层分组
    - 不同控件组成多个平行窗格，只有一个层次深度
- PPT的Option
  - 标签堆叠成两行
- 建议
  - 不要堆叠标签



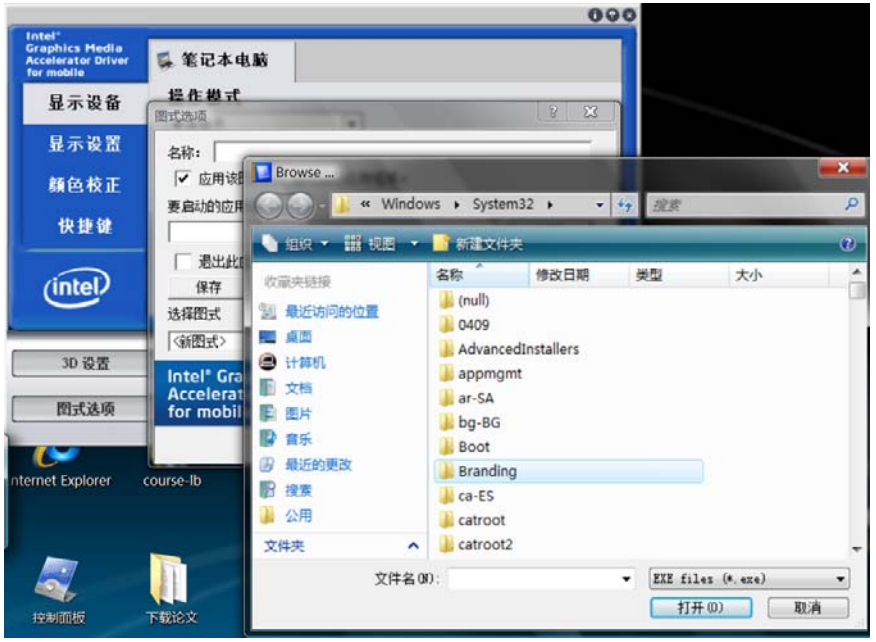


- 扩展对话框
  - 1990年左右盛行
  - 新手用户不必面对复杂的工具，熟练用户也不必为寻找这些工具烦恼



- 注意
  - 必须小心设计
  - 否则对初学者傲慢无礼，专家使用也有麻烦
  - 最好记住上次被调用时所处的使用状态

- 级联对话框
  - 糟糕的习惯用法
  - 一个对话框的控件在一个层次关系的嵌套中调用另一个对话框
- 分析
  - 适合处理深度问题，但层次太深导致界面复杂



对话框设计原则

- 把主要的交互操作放在主窗口中
  - 对话框适合主交互流之外的功能
- 视觉上区分模态与非模态对话框
  - 为非模态对话框提供一致的终止命令
- 不要用临时对话框作为错误信息框或确认信息框
  - 保证用户能够阅读
- 不要堆叠标签

主要内容

- 窗口和菜单
- 对话框
- 常用控件
- 工具栏
- 屏幕复杂性度量
- 用户界面设计原理

控件

- 使用者和数字产品进行交流的屏幕对象
  - 具有可操作性和自包含性
- 控件的使用必须恰当且合理
  - 大多数布满控件的对话框都不是好的用户界面设计
- 根据用户目标，控件可分为4种基本类型
  - 命令控件
  - 选择控件
  - 显示控件
  - 文本输入控件

命令控件

- 接收操作并立即执行
- 举例：按钮

  - 单击释放后立即执行
  - “可按压特性”，点击时视觉发生改变
    - 不改变会使用户不安，“到底被按下了吗？”
  - 图标按钮（butcons）：工具栏中的按钮

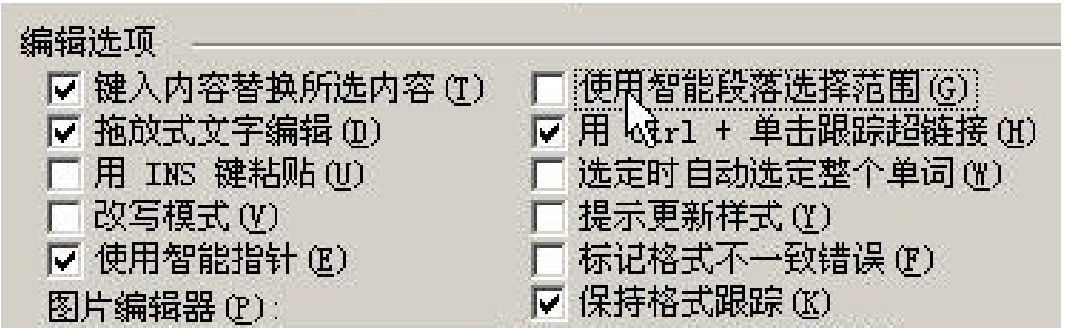
- 矩形->方形，有文字->有图形，有突起->无突起

选择控件

- 允许用户从一组有效的选项中选择一个操作数，还能用来设定操作
  - 也可用来定义形容词或副词
  - 部分情况下，选择控件也可以触发操作
    - 如使用下拉列表调整字体大小等，较为自然
    - 网页中下拉列表触发操作有可能把用户搞糊涂
- 常见选择控件
  - 复选框
  - 列表框
  - 下拉列表

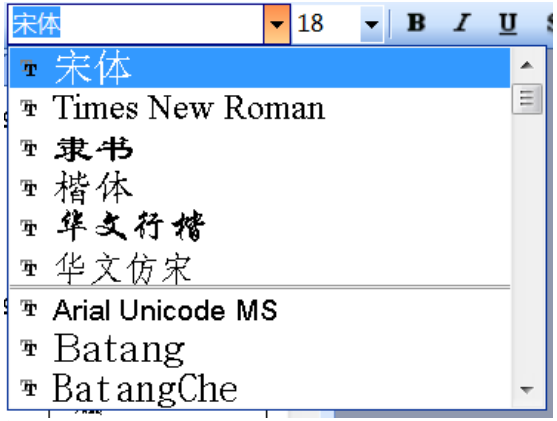
复选框

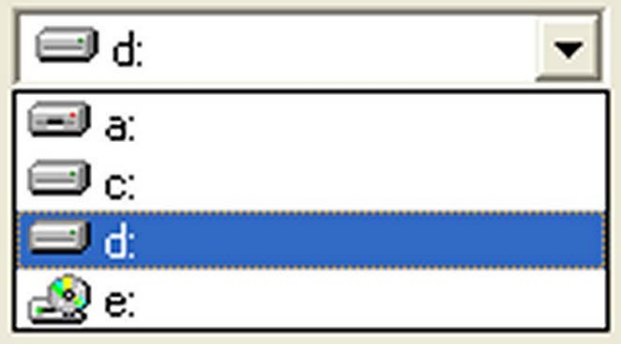
- 简单、可见、优雅
- 确切的文本使复选框清楚明确
- 但用户不得不放慢阅读速度
- 占据了数量可观的屏幕空间
- 方形复选框已经成为一个重要的标准
- 变体：单选按钮



列表框

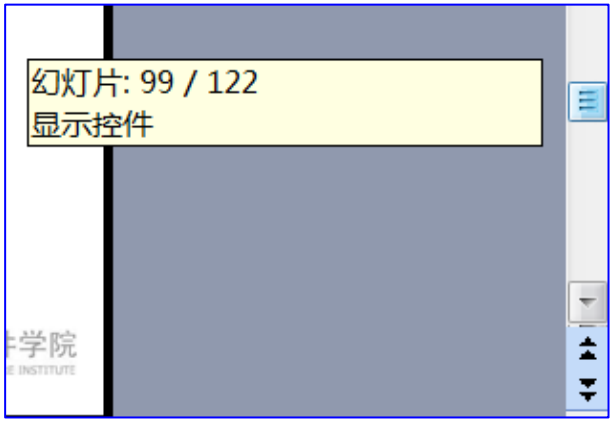
- 允许用户从有限的文本字符串中选择，每个文本字符串代表一个命令、对象或属性
  - 又称“列表框”或“列表视图”
- 变体
  - 下拉列表、列表中的图标、预览视图
- 标准的选择模态是相互排斥的
  - 选择一个事物时，先前选定的事物就会取消





显示控件

- 显示和管理屏幕上信息的视觉显示方式
  - 如管理对象在屏幕上的视觉显示方式的控件及静态显示只读信息的控件等



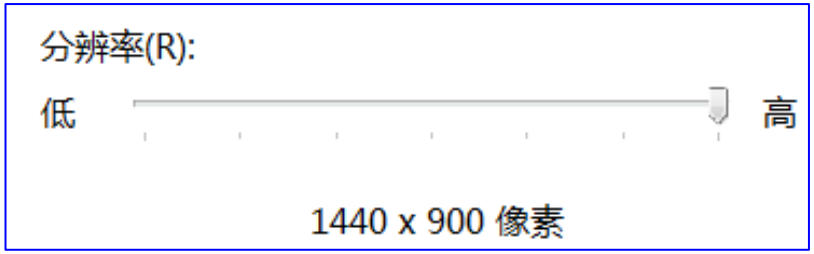
- 滚动条
  - 适合用于窗口内容和文本导航器
  - 包含的信息
    - 当前可见文档的百分比
    - 总共有多少页
    - 拖动滑块时，显示页数
    - 拖动滑块时，显示每一页的第一个句子
    - 跳到文档开始和末尾的按钮

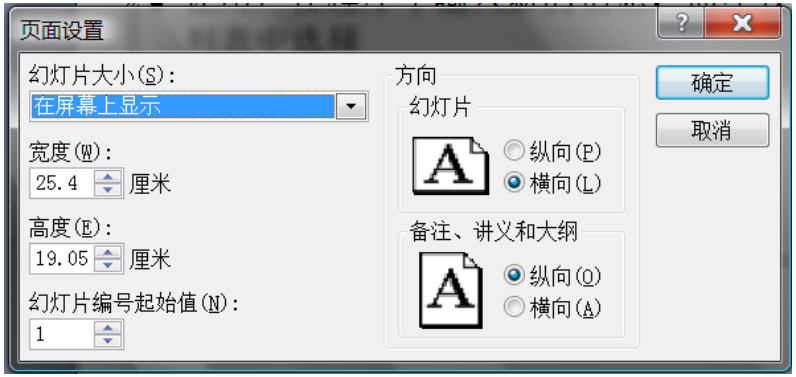
讨论：滚动条的滥用

- 日历中的滚动条有必要吗？
  - 时间是没有真正意义上的开始和结束的
  - 不适合为无限时间轴进行导航

输入控件

- 让用户在程序中输入新的信息，而不仅仅是从列表中选择
- 无界输入可能导致十分严重的问题
- 有界输入控件
  - 用于在任何需要有数值界限的地方
    - 杜绝“实际上不能却说可能”的粗鲁方式
  - 举例：滑动条、微调控制项、标尺等





- 无界输入控件
  - 允许用户输入任意文本值，包括文字和数字
  - 如文本编辑控件
  - 某些情况下，可接收值的集合是有限的，但是对列表控件来说数量太大
  - 输入数据后加以验证
    - 验证控件
    - 为用户提供充分反馈
  - 保留输入的有效信息

主要内容

- 窗口和菜单
- 对话框
- 常用控件
- 工具栏
- 屏幕复杂性度量
- 用户界面设计原理

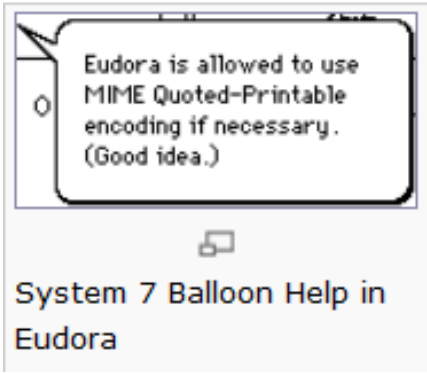
工具栏

- 微软第一个将其引入用户界面
- 工具栏 VS. 菜单
  - 都提供对程序功能的访问
  - 菜单提供完整的工具集，主要用于教学
  - 工具栏是为经常使用的命令设置的，对新手用户帮助不大
    - 工具提示可以在一定程度上缓解这个问题
  - “工具栏是单行(或单列)排列且始终可见的图形化立即菜单项”

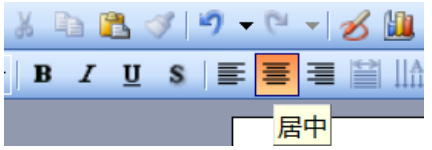
- 工具栏上的图标与文本
  - 文本标签精确，但阅读与识别速度较慢
  - 工具栏和菜单的用途不同
    - 工具栏主要为常用功能提供快速访问
    - 图形的表意特征较文本更适合担当这种角色
  - 图标图像
    - 找代表事物的图像比寻找代表动作或关系的图像容易得多，如垃圾桶、打印机等
- Q:为什么不能同时使用文本和图像？
    - 屏幕空间非常宝贵
    - 解决：工具提示

解释工具栏控件

- Apple公司的System 7操作系统最早尝试解决该问题：气球帮助
  - 鼠标经过与气球弹出之间没有延迟
  - 气球会遮挡一大块应用的区域
  - 模态帮助系统，无法同时学习和使用

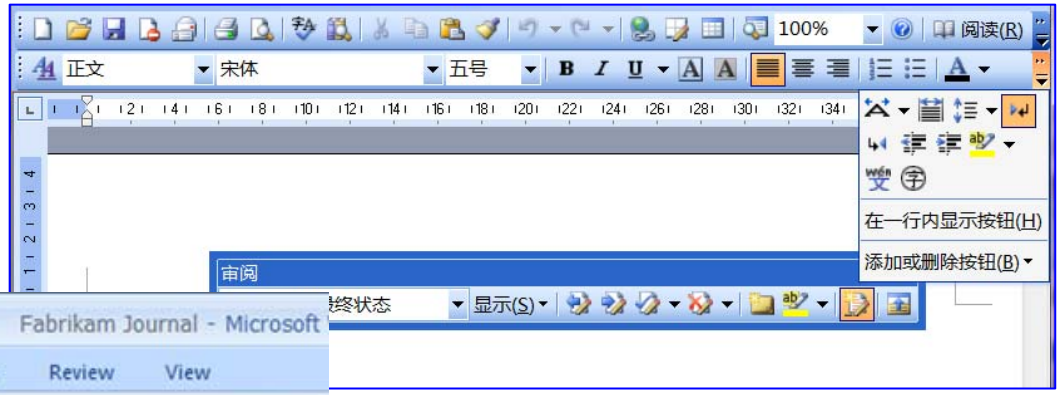


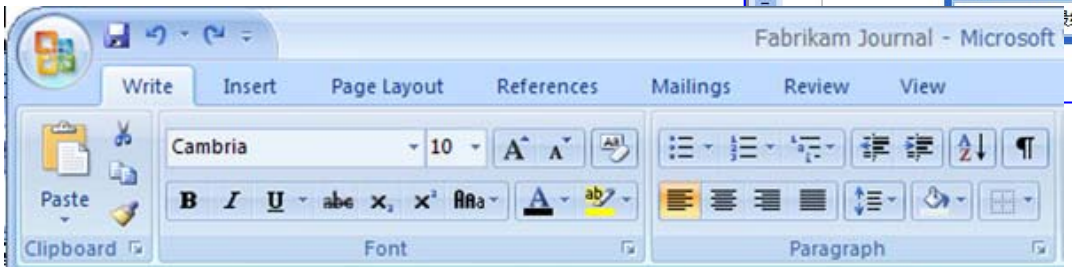
- 工具提示(ToolTips)
  - Microsoft提出的气球帮助的变体
  - 延时出现的时机非常好，大约1秒后显示帮助
  - 只包含一个单词或非常短的词组



工具栏的演化

- 没有理由将工具栏控件限制为图标按钮
  - 组合框：Word样式、字体和字号控制
  - 状态指示
  - 工具栏上的菜单
  - 可移动工具栏
  - 可定制工具栏
  - 带条





- 屏幕复杂性度量

布局复杂度

- 根据可视对象的大小和位置来衡量
  - 如果可视对象在高度和宽度上经常改变以及对象与可视交互环境边界之间的距离比较大的话，就可以说这个布局较复杂

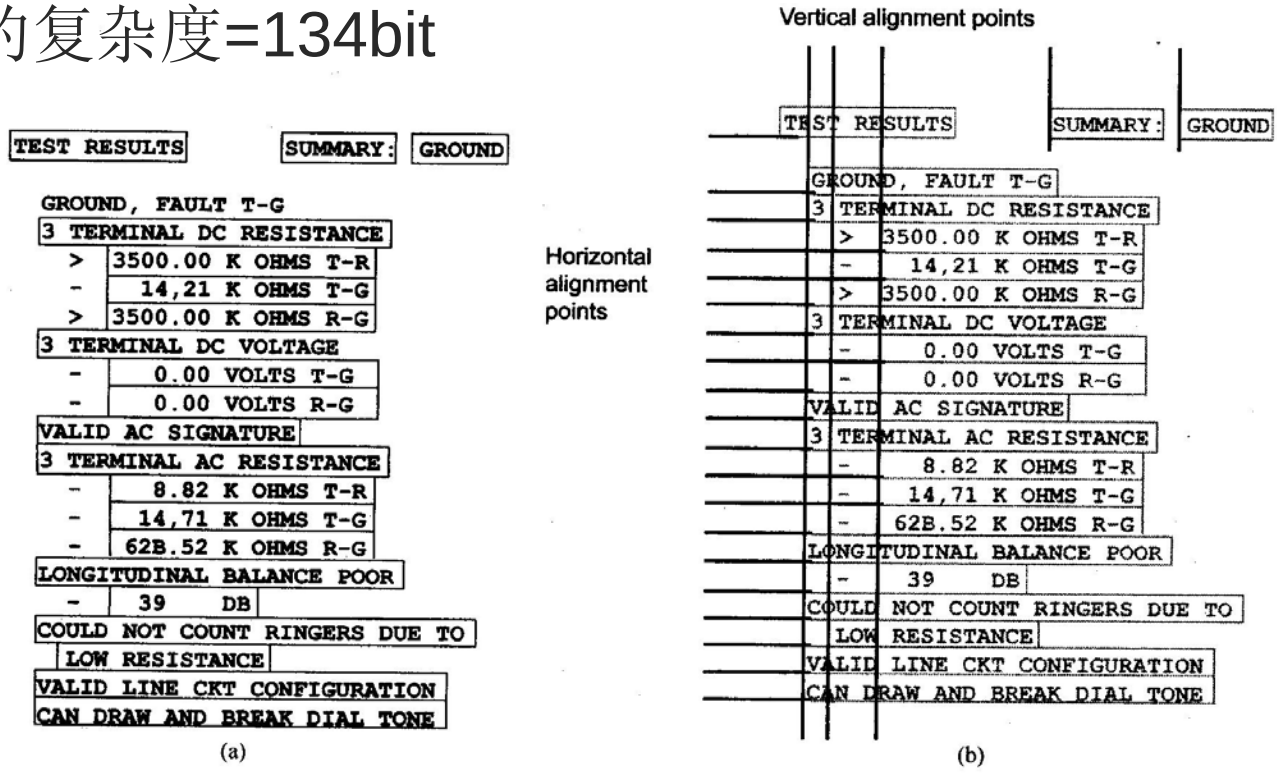
$$C = -N\sum_{n=1}^{m} p_n \log_2 p_n$$

  - C：以比特表示的系统复杂性；N：所有组件的数量
  - m: 组件分类的数目；pn：第n类组件出现的概率（以该类组件出现的频率为基础）

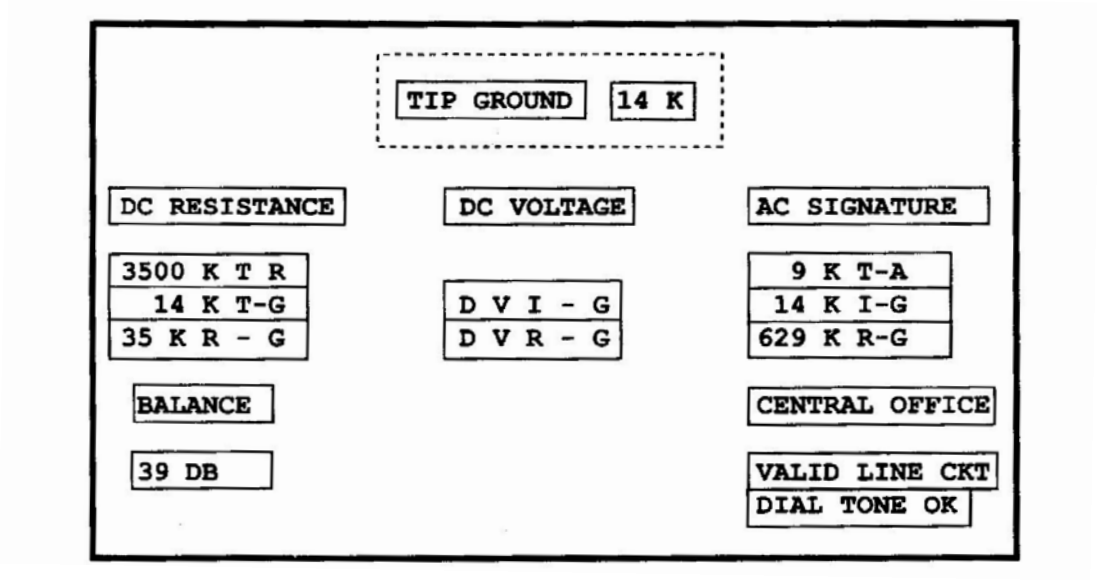
计算步骤

- 使用矩形包围屏幕中的每个元素
- 计算屏幕中元素的数目及列的数目（垂直方向上的对齐点）
- 计算元素的数目及行的数目（水平方向上的对齐点）

- 22个元素和6个垂直方向上的对齐点 = 41 bit
- 22个元素和20个水平方向上的对齐点 = 93 bit
- 整体的复杂度=134bit



- 18个元素和7个垂直方向上的对齐点 = 43 bit
- 18个元素和8个水平方向上的对齐点 = 93 bit
- 整体的复杂度 = 96 bit
  - 与原屏幕相比节省了28%的复杂度



改进算法

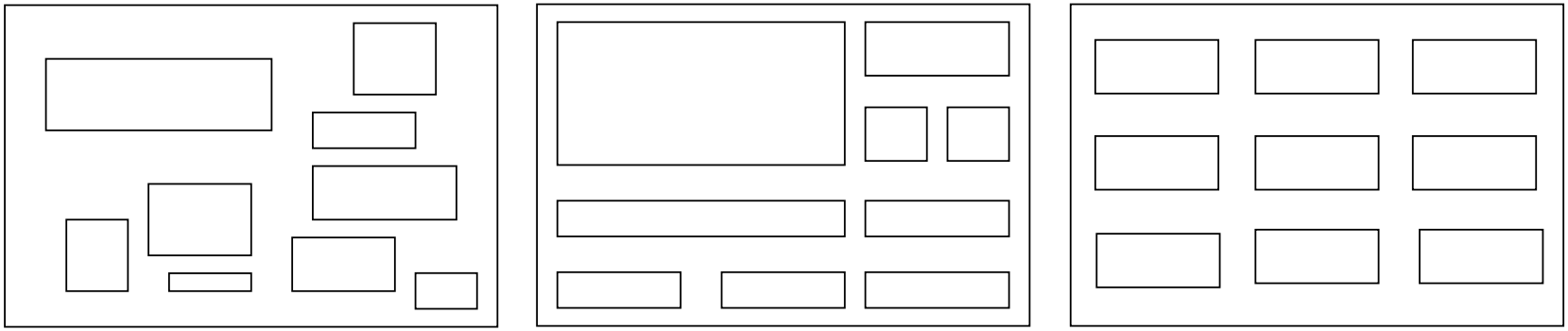
- 只需把可视组件总数加上顶端未对齐组件的个数和左侧未对其组件的个数，就可以得到相同的最终结果, by Galitz
  - 屏幕中元素的个数
  - 水平对齐点的个数
  - 垂直对齐点的个数
- 过分简单的接口会降低其实用性
  - 具有中等布局复杂度的设计更好一些

- 原屏幕：
  - 22个元素
  - 6个水平对齐点
  - 20个垂直对齐点
  - 复杂度为48
- 重新设计后的屏幕：
  - 18个元素
  - 7个水平对齐点
  - 8个垂直对齐点
  - 复杂度为33

布局统一度

- 布局复杂度
  - 是严格的结构度量指标
  - 不考虑可视组件位于何处，只考虑组件在大小上的变化及其与边界之间的距离
  - 在规划和改进真实设计的指南的使用范围相对有限
- 布局统一度
  - 只对界面组成部分的空间排列进行衡量
  - 不考虑这些组成部分是什么以及如何使用
  - 以“视觉上无序的排列有碍于可用性”这一原理为基础

- 当可视组件整齐排列或者组件尺寸相差不大时，布局统一度就会提高
  - 左：各组件在大小或位置上没有一致性，布局统一度是0%
  - 右：各组件的布局和大小完全一致，该接口布局统一度是100%
  - 中庸的设计，布局统一度82.5%
- LU介于50%和85%之间时是比较合理的



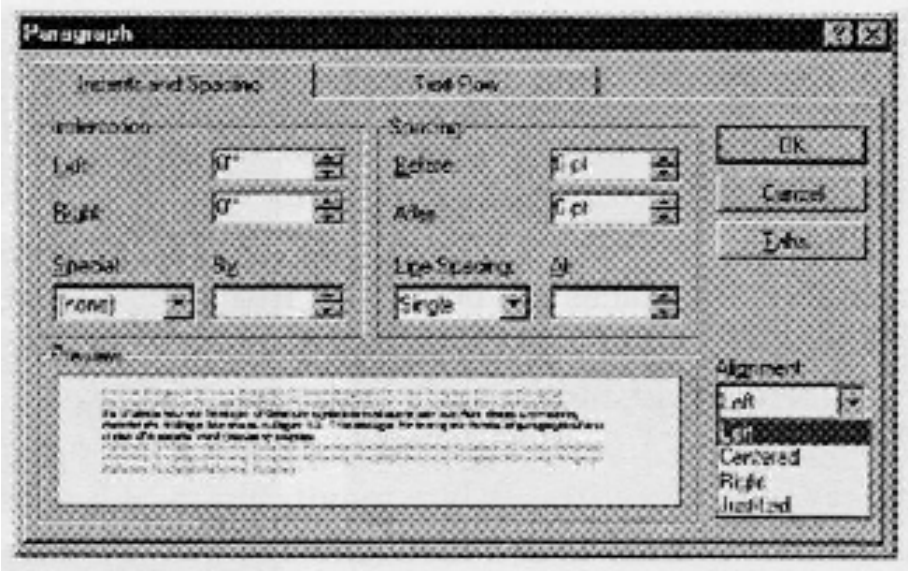
用户界面设计原理

结构原理

根据清楚而一致的模型，以一种有意义和有用的方式对用户接口进行组织，把相关的东西放在一起，把不相关的东西分开放，区别不同的东西，使类似的东西看起来相似。

- 好的用户界面设计对用户界面的组织方式应符合所支持工作的结构，符合用户对工作的认识
- 过滥地运用隐喻会让用户界面难以理解

- 正文左对齐、右对齐和居中
  - 使用垂直安排是一种不合适的可视隐喻
- 缩进和边界对齐都与正文的边界有关
  - 但这些在视觉上和逻辑上相关的功能却在空间上被分开安排



简单性原理

使简单、常用的功能简便易行，用用户自己的语言进行简明易懂的交流，对冗长的操作过程提供与其语义相关的快捷方式。

- 应从用户的角度来看什么任务更常用、什么任务更简单
- 并不一定非要进行调查获广泛研究
- 软盘格式化实用程序举例
  - 没有考虑最简单、最常用和最核心的任务究竟是什么
  - 让必要的确认和次要功能干扰了本来很简单的常用任务的顺利执行

可见性原理

让完成给定用户所需的所有选项和材料对用户可见，不要让额外或冗余的信息干扰用户。

- 在“所见即所得”的基础上，进一步实现“所见即所需”
  - 设计目标是让所有需要和相关的选项可见和明确
  - 不应让用户被不必要的信息所迷惑
- 传真机管理程序举例
  - 功能的取名以及在用户界面上的位置安排存在问题
  - 是一个由内向外的设计


Fax Log Send Receive Setup
Services Window Help
Send
Send Log
Rcv Log
Attach
Send Lo
Date
Time
Pages
Option
05-Apr-95
07:15:22
05-Apr-95
07:14:46
04-Apr-95
29-Mar-95
02:27:42
29-Mar-95
02:26:40
Recipient
Ready

反馈原理

通过用户所熟悉的清楚、简洁和无歧义的语言，让用户时刻了解系统对用户操作的反应和解释，了解与用户有关且被他们所关心的系统状态变化、出错、异常等所有情况。

- 成功的反馈就是以能被对方注意到、读到和正确理解的方式来提供信息
  - 屏幕中央和顶端出现的反馈容易被注意
  - 屏幕底端是最不容易被注意到的
- 好的出错信息
  - 以一个能让用户马上知道问题何在的题头开始
  - 有针对性和简洁地解释问题究竟是什么
  - 建议解决问题的方法或操作步骤

宽容原理

保持灵活和宽容，提供通过撤销和重做功能来减少用户出错和不当操作所带来的开销，同时保持允许各种不同的输入形式和顺序以及通过合理地解释用户的所有合理操作来尽可能防止出错。

- 高可用性的系统帮助用户少犯错误
- 宽容性如何取决于将什么数据检测为有效以及何时进行这种合法性检查
  - 检查后，在返回给用户的屏幕上将所有不合法的数据域置为高亮度，把光标放在第一个出错的数据域，并且在状态栏中给出解释的话
  - 过分使用合法性检查会降低可用性

重用原理

对内部和外部的组件和行为加以重用，有目的而不是无目的地维持一致性，从而减少用户重新思考和记忆的需要。

- 用户界面内在外观、位置以及行为上的一致性使得软件容易学习和记忆如何使用
  - 界面更具有可预测性和更容易理解
- 重用程度越高，一致性就越好
  - 不一致的用户界面不仅会降低软件的可用性，而且会增加程序设计的工作量

小结

- 窗口
- 对话框
- 控件
- 菜单
- 工具栏
- 错误、警告和确认
- 屏幕复杂度度量
- 用户界面设计原理