# Ch.14 Component-Level Design

May 18, 2015

## What is a Component?

**"a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces."**

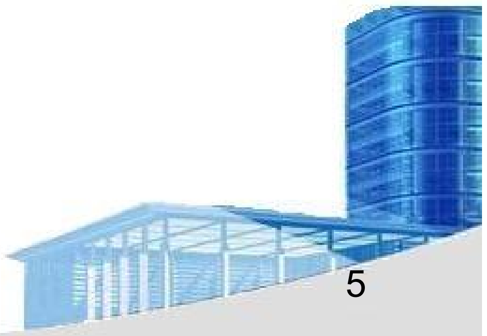
**——*OMG Unified Modeling Language Specification* [OMG01]**

- *OO view*: a component contains a set of collaborating classes
- *Conventional view*: a component contains processing logic, the internal data structures that are required to implement the processing logic, and an interface that enables the component to be invoked and data to be passed to it.

- **Basic design principles**
  - **The Open-Closed Principle (OCP).** *“A module [component] should be open for extension but closed for modification.*
  - **The Liskov Substitution Principle (LSP).** *“Subclasses should be substitutable for their base classes.*
  - **Dependency Inversion Principle (DIP).** *“Depend on abstractions. Do not depend on concretions.”*
  - **The Interface Segregation Principle (ISP).** *“Many client-specific interfaces are better than one general purpose interface.*
  - **The Release Reuse Equivalency Principle (REP).** *“The granule of reuse is the granule of release.”*
  - **The Common Closure Principle (CCP).** *“Classes that change together belong together.”*
  - **The Common Reuse Principle (CRP).** *“Classes that aren’t reused together should not be grouped together.”*

- **Design Guidelines**
  - *Components*
    - Naming conventions(命名约定) should be established for components that are specified as part of the architectural model and then refined and elaborated as part of the component-level model. Ex. FloorPlan
  - *Interfaces*
    - Interfaces provide important information about communication and collaboration (as well as helping us to achieve the OCP)
  - *Dependencies and Inheritance*
    - it is a good idea to model dependencies from left to right and inheritance from bottom (derived classes) to top (base classes).

- **Cohesion（内聚性）**
  - Conventional view:
    - the “single-mindedness” (专诚性、单一性)of a module
  - O-O view:
    - cohesion implies that a component or class encapsulates only attributes and operations that are closely related to one another and to the class or component itself
  - Levels of cohesion
    - Functional
    - Layer
    - Communicational
    - Sequential
    - Procedural
    - Temporal
    - Utility(功用)



## Coupling

- Conventional view:
  - The degree to which a component is connected to other components and to the external world
- OO view:
  - a qualitative measure of the degree to which classes are connected to one another
- Level of coupling
  - Content
  - Common
  - Control
  - Stamp
  - Data
  - Routine call
  - Type use
  - Inclusion or import
  - External

## Component Level Design - I

- Step 1. Identify all design classes that correspond to the problem domain.
- Step 2. Identify all design classes that correspond to the infrastructure domain.
- Step 3. Elaborate all design classes that are not acquired as reusable components.
- Step 3a. Specify message details when classes or component collaborate.
- Step 3b. Identify appropriate interfaces for each component.

## • Component Level Design - II

- Step 3c. Elaborate attributes and define data types and data structures required to implement them.
- Step 3d. Describe processing flow within each operation in detail.
- Step 4. Describe persistent data sources (databases and files) and identify the classes required to manage them.
- Step 5. Develop and elaborate behavioral representations for a class or component.
- Step 6. Elaborate deployment diagrams to provide additional implementation detail.
- Step 7. Factor every component-level design representation and always consider alternatives.

- **Component Design for WebApps**

  - WebApp component is
    - (1) a well-defined cohesive function that manipulates content or provides computational or data processing for an end-user, or
    - (2) a cohesive package of content and functionality that provides end-user with some required capability.
  - Therefore, component-level design for WebApps often incorporates elements of content design and functional design.

## • Content Design for WebApps

- focuses on content objects and the manner in which they may be packaged for presentation to a WebApp end-user

---consider a Web-based video surveillance capability within SafeHomeAssured.com

- – potential content components can be defined for the video surveillance capability:
  - – (1) the content objects that represent the space layout (the floor plan) with additional icons representing the location of sensors and video cameras;
  - – (2) the collection of thumbnail video captures (each an separate data object), and
  - – (3) the streaming video window for a specific camera.
- – Each of these components can be separately named and manipulated as a package.

## Functional Design for WebApps

- Modern Web applications deliver increasingly sophisticated processing functions that:
  - (1) perform localized processing to generate content and navigation capability in a dynamic fashion;
  - (2) provide computation or data processing capability that is appropriate for the WebApp's business domain;
  - (3) provide sophisticated database query and access, or
  - (4) establish data interfaces with external corporate systems.
- To achieve these (and many other) capabilities, you will design and construct WebApp functional components that are identical in form to software components for conventional software.

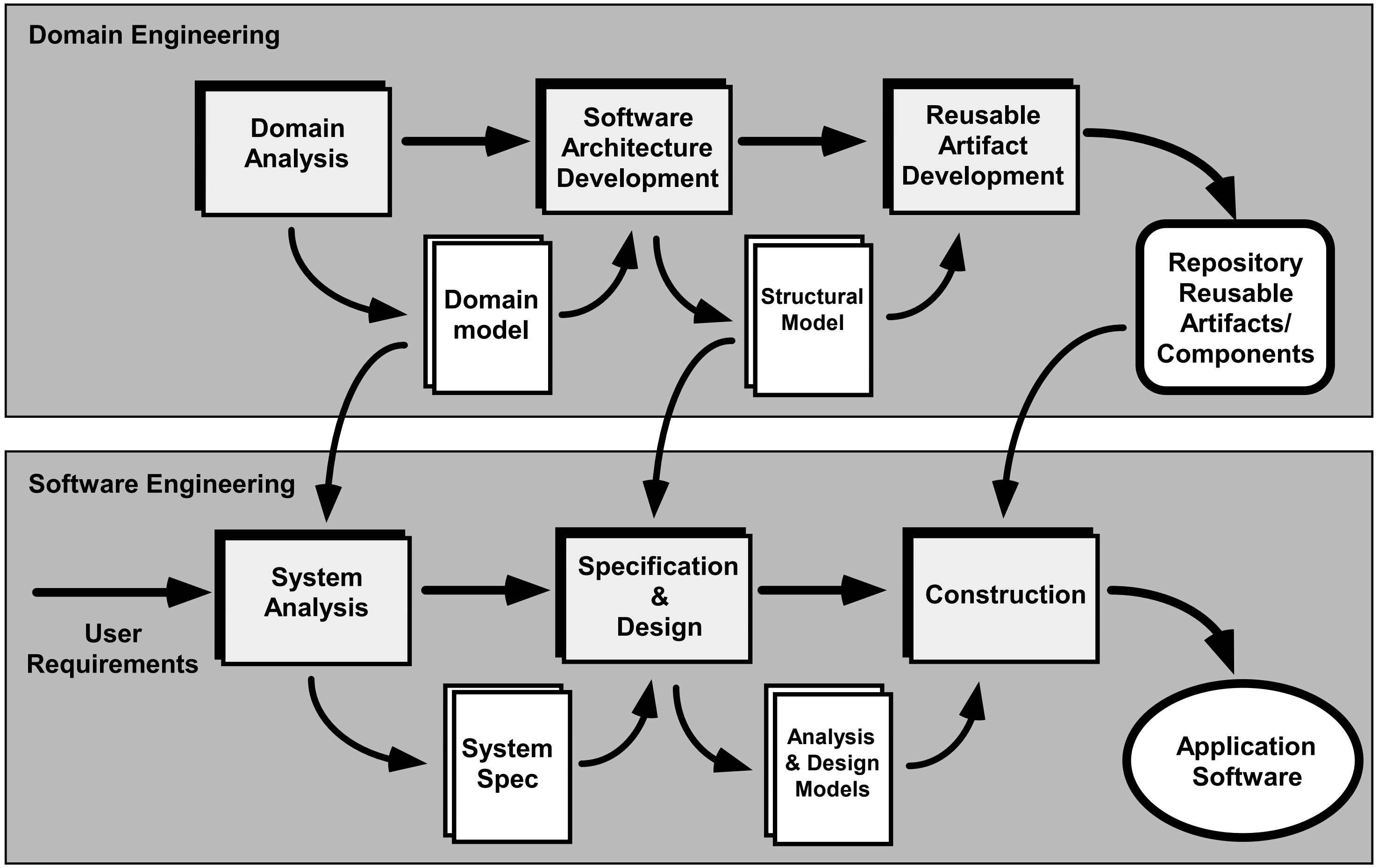
- **Designing Conventional Components**

  - The design of processing logic is governed by the basic principles of algorithm design and structured programming
  - The design of data structures is defined by the data model developed for the system
  - The design of interfaces is governed by the collaborations that a component must effect.

- **Component-Based Development**

  - When faced with the possibility of reuse, the software team asks:
    - Are commercial off-the-shelf (COTS) components available to implement the requirement?
    - Are internally-developed reusable components available to implement the requirement?
    - Are the interfaces for available components compatible within the architecture of the system to be built?
  - At the same time, they are faced with some impediments(障碍) to reuse ...

- **The CBSE Process** （Component Based Software Engineering）



- **Domain Engineering**
  - 1. Define the domain to be investigated.
  - 2. Categorize the items extracted from the domain.
  - 3. Collect a representative sample of applications in the domain.
  - 4. Analyze each application in the sample.
  - 5. Develop an analysis model for the objects.

## Identifying Reusable Components

- Is component functionality required on future implementations?
- How common is the component's function within the domain?
- Is there duplication of the component's function within the domain?
- Is the component hardware-dependent?
- Does the hardware remain unchanged between implementations?
- Can the hardware specifics be removed to another component?
- Is the design optimized enough for the next implementation?
- Can we parameterize a non-reusable component so that it becomes reusable?
- Is the component reusable in many implementations with only minor changes?
- Is reuse through modification feasible?
- Can a non-reusable component be decomposed to yield reusable components?
- How valid is component decomposition for reuse?

- **Component-Based SE （CBSE）**
  - a library of components must be available
  - components should have a consistent structure
  - a standard should exist, e.g.,
    - OMG/CORBA
    - Microsoft COM
    - Sun JavaBeans

## • CBSE Activities

-

- Component qualification
- Component adaptation
- Component composition
- Component update

## • Qualification

- *Mitch Kapor, the creator of Lotus 1-2-3, presented a "software design manifesto". He said:*
  - application programming interface (API)
  - development and integration tools required by the component
  - run-time requirements including resource usage (e.g., memory or storage), timing or speed, and network protocol
  - service requirements including operating system interfaces and support from other components
  - security features including access controls and authentication protocol
  - embedded design assumptions including the use of specific numerical or non-numerical algorithms
  - exception handling

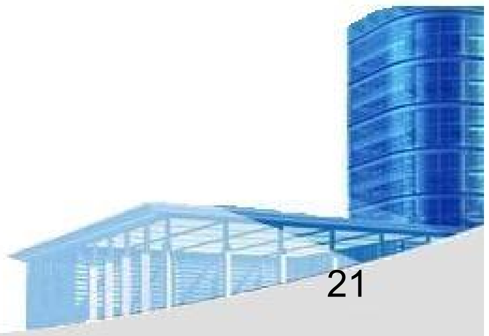
- **Adaptation**
  - *The implication of "easy integration" is:*

    (1) that consistent methods of resource management have been implemented for all components in the library;

    (2) that common activities such as data management exist for all components, and

    (3) that interfaces within the architecture and with the external environment have been implemented in a consistent manner.

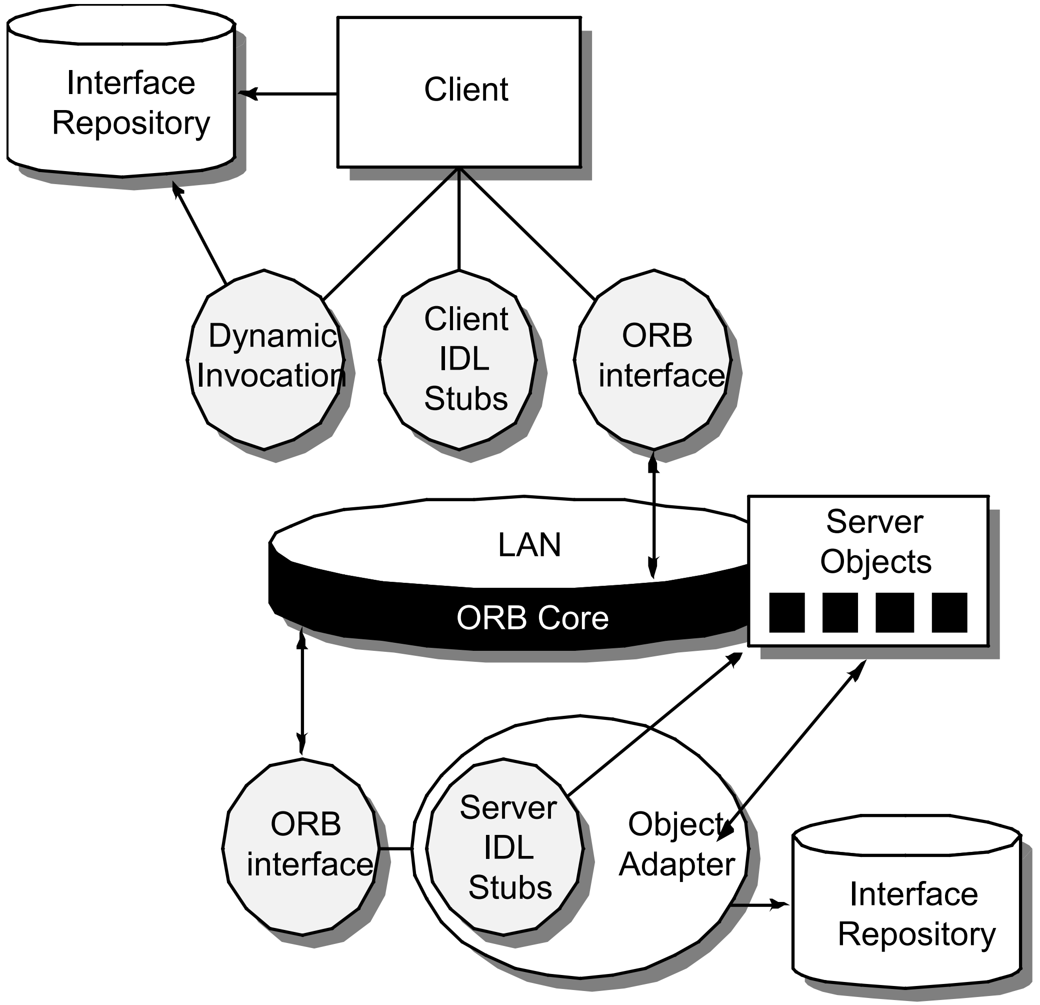
- **Composition**
  - An infrastructure must be established to bind(结合) components together
  - Architectural ingredients for composition include:
    - Data exchange model
    - Automation
    - Structured storage
    - Underlying object model



## OMG/ CORBA

- The Object Management Group has published a *common object request broker architecture* (OMG/**CORBA**).
- An object request broker (ORB) provides services that enable reusable components (objects) to communicate with other components, regardless of their location within a system.
- Integration of CORBA components (without modification) within a system is assured if an interface definition language (IDL) interface is created for every component.
- Objects within the client application request one or more services from the ORB server. Requests are made via an IDL or dynamically at run time.
- An interface repository contains all necessary information about the service's request and response formats.

## • ORB Architecture



## Microsoft COM

- The component object model (COM) provides a specification for using components produced by various vendors within a single application running under the Windows operating system.
- COM encompasses two elements:
  - COM interfaces (implemented as COM objects)
  - a set of mechanisms for registering and passing messages between COM interfaces.

## • Sun JavaBeans

- The JavaBeans component system is a portable, platform independent CBSE infrastructure developed using the Java programming language.
- The JavaBeans component system encompasses a set of tools, called the Bean Development Kit (BDK), that allows developers to
  - analyze how existing Beans (components) work
  - customize their behavior and appearance
  - establish mechanisms for coordination and communication
  - develop custom Beans for use in a specific application
  - test and evaluate Bean behavior.

- **The Reuse Environment**
  - A component database capable of storing software components and the classification information necessary to retrieve(恢复) them.
  - A library management system that provides access to the database.
  - A software component retrieval system (e.g., an object request broker) that enables a client application to retrieve(恢复/重新得到) components and services from the library server.
  - CBSE tools that support the integration of reused components into a new design or implementation.

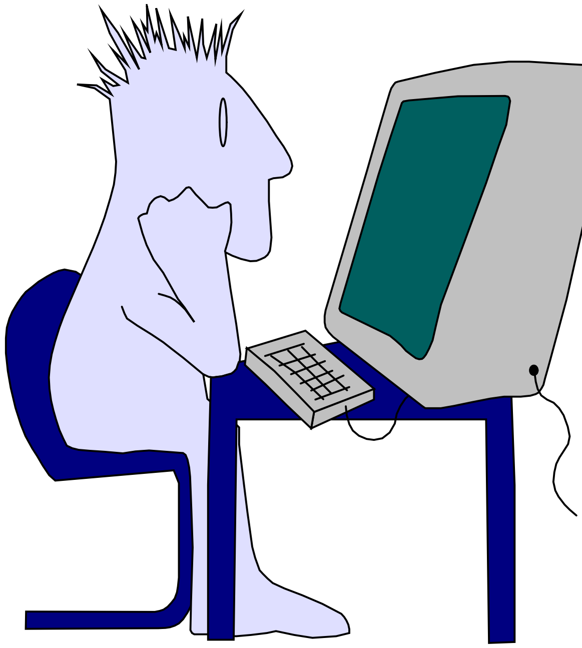
# Ch.15  User Interface Design

## • Interface Design

- Easy to learn?
  - Easy to use?
    - Easy to understand?

- **Interface Design**

  - *Typical Design Errors*
    - lack of consistency
    - too much memorization
    - no guidance / help
    - no context sensitivity
    - poor response
    - **Arcane**（晦涩难解的）/unfriendly



## Three Golden Rules

- ☞ Place the user in control
- ☞ Reduce the user's memory load
- ☞ Make the interface consistent

- **Place the User in Control**
  - Define interaction modes in a way that does not force a user into unnecessary or undesired actions.
  - Provide for flexible interaction.
  - Allow user interaction to be interruptible and undoable.
  - Streamline interaction as skill levels advance and allow the interaction to be customized.
  - Hide technical internals from the casual user.
  - Design for direct interaction with objects that appear on the screen.

- **Reduce the User’s Memory Load**
  - Reduce demand on short-term memory.
  - Establish meaningful defaults.
  - Define shortcuts that are intuitive.

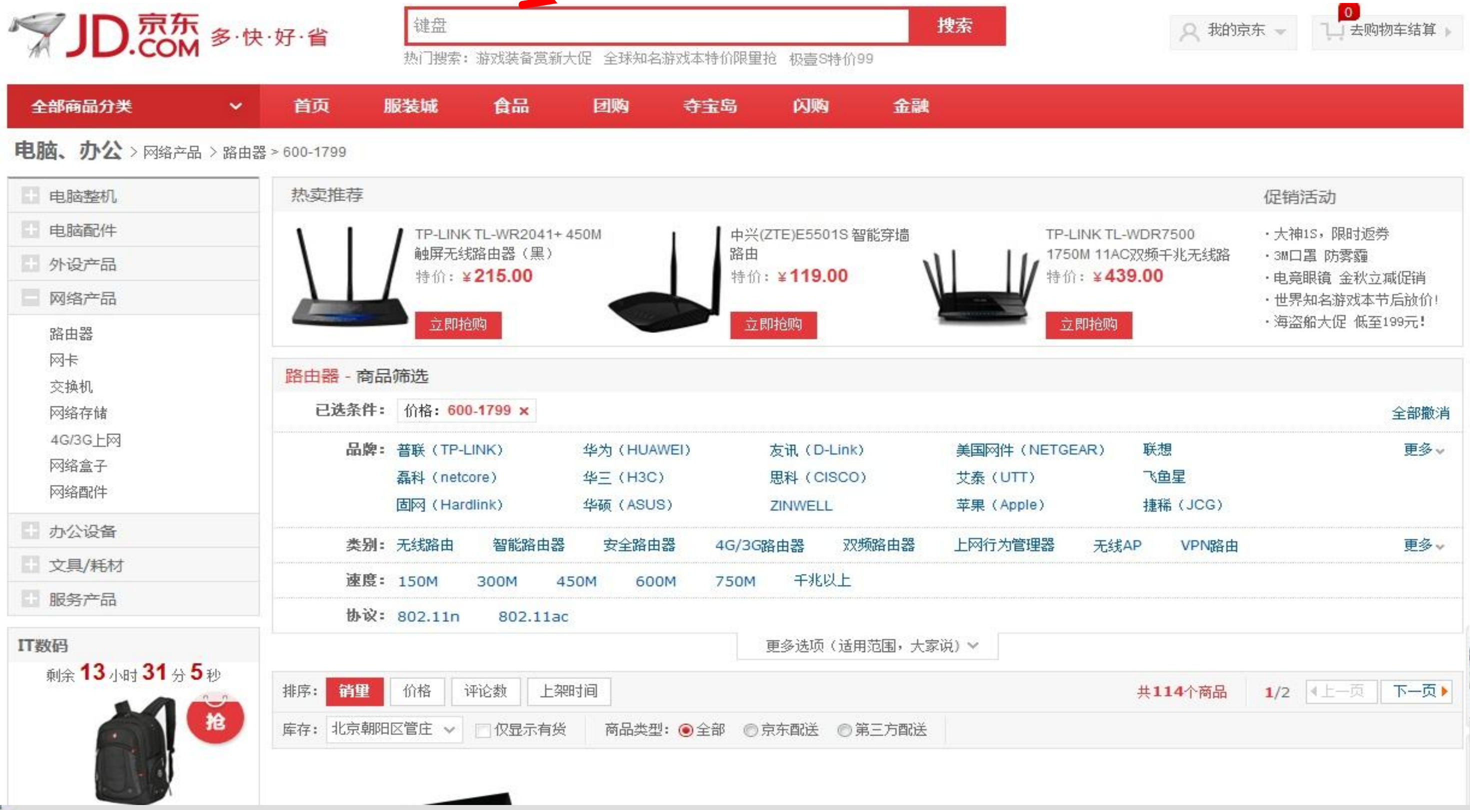
    (e.g. Alt + P;  Ctrl + Z, X, C, V)
  - The visual layout of the interface should be based on a real world metaphor.
  - Disclose information in a progressive fashion.

## Make the Interface Consistent

- Allow the user to put the current task into a meaningful context.
- Maintain consistency across a family of applications.
- If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. Ex. 粘贴(Paste)：Ctrl+P vs Ctrl+V

## • Make the Interface Consistent

- Allow the user to put the current task into a meaningful context.



JD.京东.COM 多·快·好·省

键盘 搜索

热门搜索：游戏装备赏新大促 全球知名游戏本特价限量抢 极壹S特价99

我的京东 去购物车结算 0

全部商品分类 首页 服装城 食品 团购 夺宝岛 闪购 金融

电脑、办公 > 网络产品 > 路由器 > 600-1799

- 电脑整机
- 电脑配件
- 外设产品
- 网络产品
  - 路由器
  - 网卡
  - 交换机
  - 网络存储
  - 4G/3G上网
  - 网络盒子
  - 网络配件
- 办公设备
- 文具/耗材
- 服务产品

IT数码

剩余 13 小时 31 分 5 秒 抢

热卖推荐

TP-LINK TL-WR2041+ 450M 触屏无线路由器（黑）
特价：¥215.00
立即抢购

中兴(ZTE)E5501S 智能穿墙路由
特价：¥119.00
立即抢购

TP-LINK TL-WDR7500 1750M 11AC双频千兆无线路
特价：¥439.00
立即抢购

促销活动

- 大神1S，限时返券
- 3M口罩 防雾霾
- 电竞眼镜 金秋立减促销
- 世界知名游戏本节后放价！
- 海盗船大促 低至199元！

路由器 - 商品筛选

已选条件： 价格：600-1799 × 全部撤消

| | | | | | | |
|---|---|---|---|---|---|---|
| 品牌： | 普联（TP-LINK） | 华为（HUAWEI） | 友讯（D-Link） | 美国网件（NETGEAR） | 联想 | 更多 |
| | 磊科（netcore） | 华三（H3C） | 思科（CISCO） | 艾泰（UTT） | 飞鱼星 | |
| | 固网（Hardlink） | 华硕（ASUS） | ZINWELL | 苹果（Apple） | 捷稀（JCG） | |

类别： 无线路由 智能路由器 安全路由器 4G/3G路由器 双频路由器 上网行为管理器 无线AP VPN路由 更多

速度： 150M 300M 450M 600M 750M 千兆以上

协议： 802.11n 802.11ac

更多选项（适用范围，大家说）

排序： 销量 价格 评论数 上架时间 共114个商品 1/2 上一页 下一页

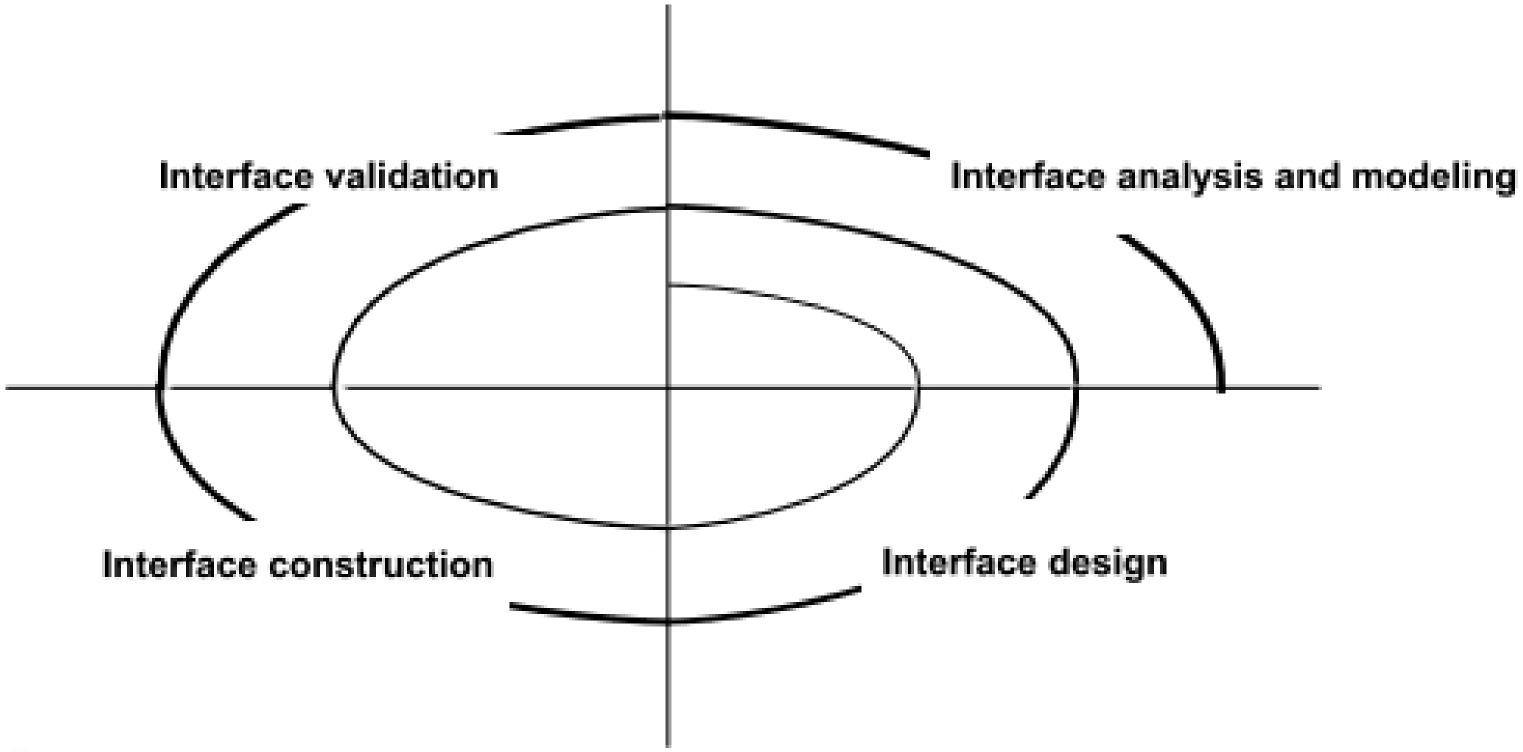
库存： 北京朝阳区管庄 ☐ 仅显示有货 商品类型： [✓ 全部] 京东配送 第三方配送

- **User Interface Analysis and Design**
  - **User model** — a profile of all end users of the system
  - **Design model** — data, architectural, interface and procedural representations of the software
  - **Mental model (system perception)** — the user's mental image of what the interface is
  - **Implementation model** — the interface "look and feel" coupled with supporting information that describe interface syntax and semantics

$\left.\right\}$ meld

***Know the user. Know the tasks.***

- **User Interface Design Process**



- **Interface Analysis**
  - Interface analysis means understanding
    - (1) the people (end-users) who will interact with the system through the interface;
    - (2) the tasks that end-users must perform to do their work,
    - (3) the content that is presented as part of the interface
    - (4) the environment in which these tasks will be conducted.

## User Analysis

- Are users trained professionals, technician, clerical, or manufacturing workers?
- What level of formal education does the average user have?
- Are the users capable of learning from written materials or have they expressed a desire for classroom training?
- Are users expert typists or keyboard phobic（病态性恐惧的）?
- What is the age range of the user community?
- Will the users be represented predominately （占优势的）by one gender?
- How are users compensated for the work they perform?
- Do users work normal office hours or do they work until the job is done?
- Is the software to be an integral part of the work users do or will it be used only occasionally?
- What is the primary spoken language among users?
- What are the consequences if a user makes a mistake using the system?
- Are users experts in the subject matter that is addressed by the system?
- Do users want to know about the technology the sits behind the interface?

## Task Analysis and Modeling

- Answers the following questions …
  - What work will the user perform in specific circumstances?
  - What tasks and subtasks will be performed as the user does the work?
  - What specific problem domain objects will the user manipulate as work is performed?
  - What is the sequence of work tasks—the workflow?
  - What is the hierarchy of tasks?
- *Use-cases* define basic interaction
- *Task elaboration* refines interactive tasks
- *Object elaboration* identifies interface objects (classes)
- *Workflow analysis* defines how a work process is completed when several people (and roles) are involved

## • Analysis of Display Content

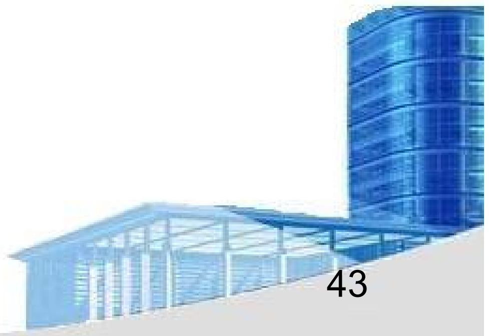
- Are different types of data assigned to consistent geographic locations on the screen (e.g., photos always appear in the upper right hand corner)?
- Can the user customize the screen location for content?
- Is proper on-screen identification assigned to all content?
- If a large report is to be presented, how should it be partitioned for ease of understanding?
- Will mechanisms be available for moving directly to summary information for large collections of data.
- Will graphical output be scaled to fit within the bounds of the display device that is used?
- How will color to be used to enhance understanding?
- How will error messages and warning be presented to the user?

## • Interface Design Steps

- Using information developed during interface analysis, *define interface objects and actions (operations).*
- *Define events (user actions)* that will cause the state of the user interface to change. Model this behavior.
- *Depict each interface state* as it will actually look to the end-user.
- *Indicate how the user interprets the state of the system* from information provided through the interface.

## • Design Issues

- Response time
- Help facilities
- Error handling
- Menu and command labeling
- Application accessibility
- Internationalization



- **Web and Mobile App Interface Design**
  - *Where am I?* The interface should
    - provide an indication of the WebApp that has been accessed
    - inform the user of her location in the content hierarchy.
  - *What can I do now?* The interface should always help the user understand his current options
    - what functions are available?
    - what links are live?
    - what content is relevant?
  - *Where have I been, where am I going?* The interface must facilitate navigation.
    - Provide a “map” (implemented in a way that is easy to understand) of where the user has been and what paths may be taken to move elsewhere within the WebApp.

- **Effective Web and Mobile App Interfaces**
  - Bruce Tognozzi [TOG01] suggests…
    - *Effective interfaces are visually apparent and forgiving*, instilling(逐步灌输) in their users a sense of control. Users quickly see the breadth of their options, grasp how to achieve their goals, and do their work.
    - *Effective interfaces do not concern the user with the inner workings of the system*. Work is carefully and continuously saved, with full option for the user to undo any activity at any time.
    - *Effective applications and services perform a maximum of work*, while requiring a minimum of information from users.

## Interface Design Principles - I

- *Anticipation*—A WebApp should be designed so that it anticipates the use's next move.
- *Communication*—The interface should communicate the status of any activity initiated by the user
- *Consistency*—The use of navigation controls, menus, icons, and aesthetics (e.g., color, shape, layout)
- *Controlled autonomy*—The interface should facilitate user movement throughout the WebApp, but it should do so in a manner that enforces navigation conventions that have been established for the application.
- *Efficiency*—The design of the WebApp and its interface should optimize the user's work efficiency, not the efficiency of the Web engineer who designs and builds it or the client-server environment that executes it.

## Interface Design Principles - II

- *Focus*—The WebApp interface (and the content it presents) should stay focused on the user task(s) at hand.
- *Fitt's Law*—"The time to acquire a target is a function of the distance to and size of the target."
- *Human interface objects*—A vast library of reusable human interface objects has been developed for WebApps.
- *Latency reduction*—The WebApp should use multi-tasking in a way that lets the user proceed with work as if the operation has been completed.
- *Learnability*— A WebApp interface should be designed to minimize learning time, and once learned, to minimize relearning required when the WebApp is revisited.

## Interface Design Principles - III

- *Maintain work product integrity*—A work product (e.g., a form completed by the user, a user specified list) must be automatically saved so that it will not be lost if an error occurs.
- *Readability*—All information presented through the interface should be readable by young and old.
- *Track state*—When appropriate, the state of the user interaction should be tracked and stored so that a user can logoff and return later to pick up where she left off.
- *Visible navigation*—A well-designed WebApp interface provides “the illusion that users are in the same place, with the work brought to them.”

## Interface Design Workflow - I

- Review information contained in the analysis model and refine as required.
- Develop a rough sketch of the Web or Mobile App interface layout.
- Map user objectives into specific interface actions.
- Define a set of user tasks that are associated with each action.
- Storyboard screen images for each interface action.
- Refine interface layout and storyboards using input from aesthetic design.

# Tasks

- **Review** Ch.14, 15;
- **Finish** “Problems and points to ponder” in Ch. 14, 15;
- **Preview** Ch. 16, 22（Testing!）, 29