Process of Interaction Design and Design Languages

Process of Interaction Design

- This week, we will explore how we can design and build interactive products
- What is different in interaction design compared to traditional software design?
  - In interaction design, we take a user-centered approach to development. This means that users' concerns direct the development rather than technical concerns.

Process of Interaction Design

- Design is also about trade-offs, or balancing conflicting requirements.
  - Generating alternatives is a principle that should be encouraged in interaction design.
  - Prototypes

Four Basic Activities of Interaction Design

- 1. Identifying needs and establishing requirements
  - In order to design something to support people, we must know who our target users are and what kind of support an interactive product could usefully provide.

Four Basic Activities of Interaction Design

- 2. Developing alternative designs
  - This is the core activity of designing: actually suggesting ideas for meeting the requirements.
  - Conceptual Design
    - Involves producing the conceptual model for the product, and a conceptual model describes what the product should do, behave, and look like.
  - Physical Design
    - Considers the detail of the product including the colors, sounds, images, menu design, and icon design.

Four Basic Activities of Interaction Design

- 3. Building interactive versions of the designs
  - The most sensible way for users to evaluate designs is to interact with them.
  - This does not mean that a software version is required, but rather, a paper-based prototype is quick and cheap to build.

Four Basic Activities of Interaction Design

- 4. Evaluating designs
  - Evaluation is the process of determining the usability and acceptability of the design.
  - Evaluation is measured in terms of a variety of criteria including:
    - numbers of errors users make using it
    - how appealing it is
    - how well it matches the requirements

Three Key Characteristics of the Interaction Design Process

- 1. A User Focus
  - A process cannot guarantee that a development will involve users, it can encourage focus on such issues and provide opportunities for evaluation and user feedback.

Three Key Characteristics of the Interaction Design Process

- 2. Specific Usability Criteria
  - Specific usability and user experience goals should be identified, clearly documented, and agreed upon and the beginning of the project.
  - They help designers choose between alternative designs and check on progress.

Three Key Characteristics of the Interaction Design Process

- 3. Iteration
  - Iteration allows designs to be refined based on feedback.
  - Iteration is important useful if you are trying to innovate. Innovation rarely emerges whole and ready to go. It takes time, evolution, trial and error, and patience.

Key Questions

- Who are the users?
- What do we mean by needs?
- How do you generate alternative designs?
- How do you chose among alternatives?

Who are the users?

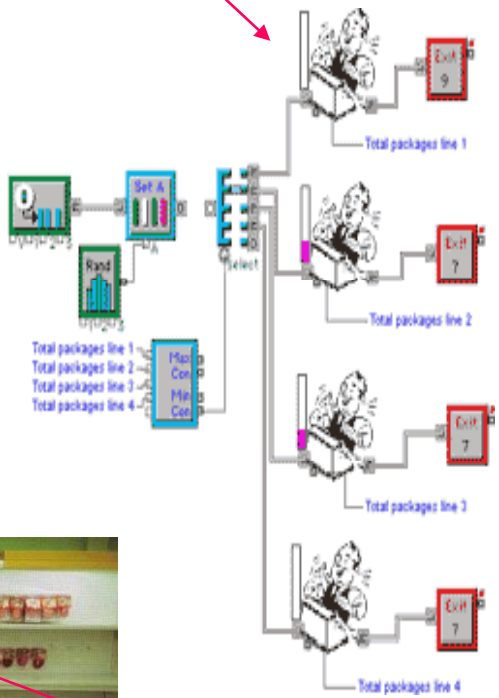
- Three types of users (Eason, 1987):
  - Primary: frequent hands-on
  - Secondary: occasional or via someone else
  - Tertiary: affected by its introduction, or will influence its purchase
- Stakeholders
  - The key persons that will be influenced by the implemented system.

Stakeholders

- Not as obvious as you think:
  - those who interact directly with the product
  - those who manage direct users
  - those who receive output from the product
  - those who make the purchasing decision
  - those who use competitor's products
- Who do you think are the stakeholders for the check-out system of a large supermarket?

Who are the stakeholders?

Check-out operators



Managers and owners

Customers

What do we mean by “needs”?

- Must understand the characteristics and capabilities of the users.
- Requires consultation from representation of target group.
- If nothing else, we should predict future behavior based on past behavior

What are ‘needs’?

- Users rarely know what is possible
- Users can’t tell you what they ‘need’ to help them achieve their goals
- Instead, look at existing tasks:
  - their context
  - what information do they require?
  - who collaborates to achieve the task?
  - why is the task achieved the way it is?
- Envisioned tasks:
  - can be rooted in existing behaviour
  - can be described as future scenarios

User Needs

- Getorade is designing a new container for it's fruit flavored sports drink. Who are the users and what would their needs be?

Generating Alternate Designs

- Creativity

Where do alternatives come from?

- Humans stick to what they know works
- But considering alternatives is important to ‘break out of the box’
- Designers are trained to consider alternatives, software people generally are not
- How do you generate alternatives?
  - ‘Flair and creativity’: research and synthesis
  - Seek inspiration: look at similar products or look at very different products

How to choose alternate design?

- Designs are external or internal
- External design is our main focus in interaction design
- Two ways to choose alternate design
  - Test the prototype, let the users choose
  - Choose what has the best “Quality”

How to choose alternate design?

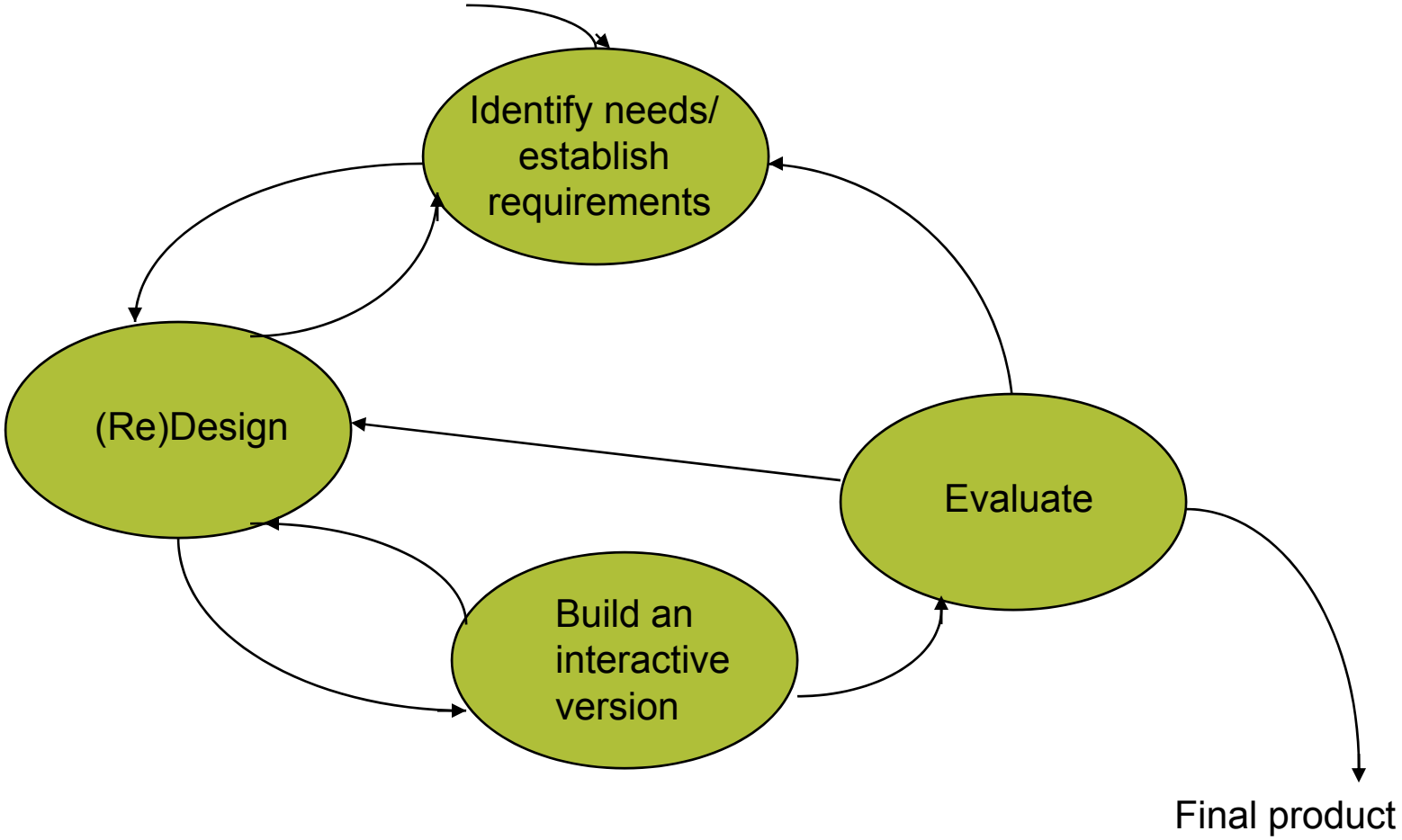
- Evaluation with users or with peers, e.g. prototypes
- Technical feasibility: some not possible
- Quality thresholds: Usability goals lead to usability criteria set early on and check regularly
  - safety: how safe?
  - utility: which functions are superfluous?
  - effectiveness: appropriate support? task coverage, information available
  - efficiency: performance measurements

Testing prototypes to choose among alternatives

Lifecycle models

- Show how the activities are related
- Lifecycle models are:
  - Management tools
  - Simplified version of reality
- Many lifecycle models exist, for example:
  - From software engineering: waterfall, spiral, JAD/RAD, Microsoft
  - From HCI: Star, usability engineering

A simple interaction design model



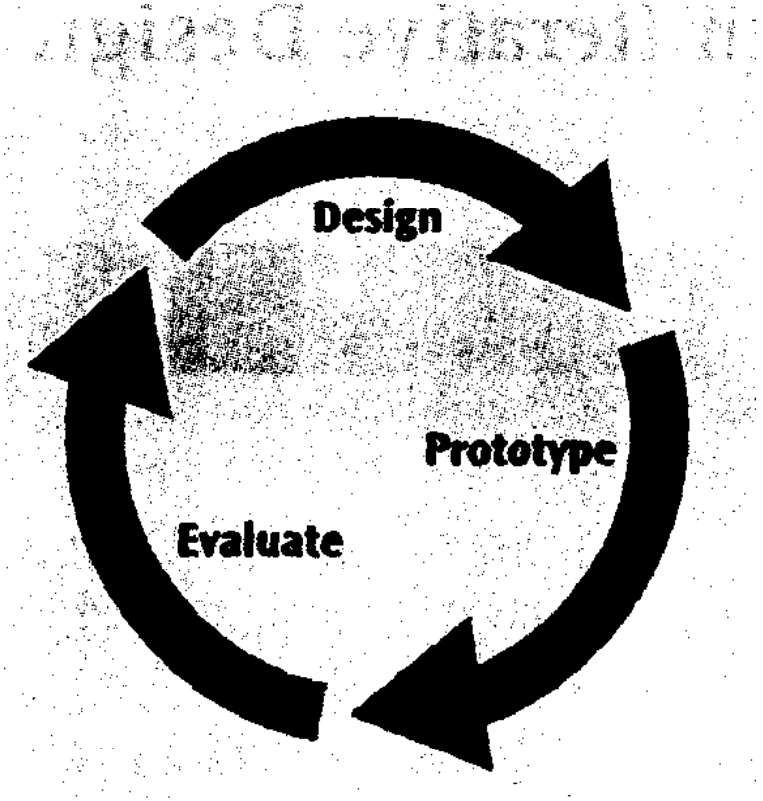
A simple interaction design model

- Most projects start with identifying needs and requirements
- From this activity, some alternative designs are generated in an attempt to meet the needs and requirements that have been identified.
- Then interactive versions of the designs are developed and evaluated
- Based on the feedback from the evaluations, the team may need to return to identifying needs or refining requirements, or it may go straight into redesigning.

A simple interaction design model

- Implicit in this cycle is that the final product will emerge in an evolutionary fashion from a rough initial idea through to the finished product
- Exactly how this evolution happens may vary from project to project.
- Development ends with an evaluation activity that ensures the final product meets the prescribed usability criteria

The Iterative Design Process

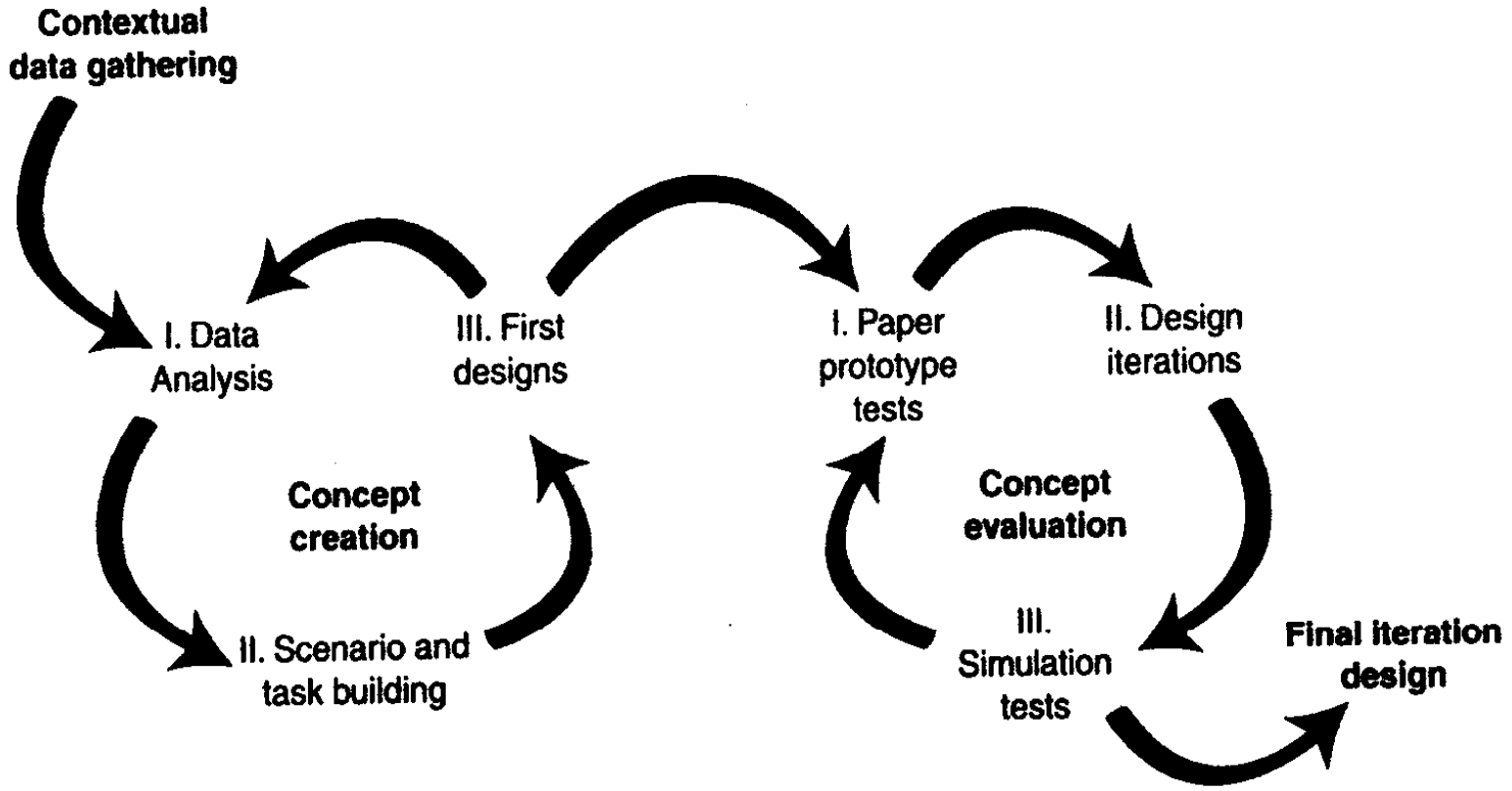


Iterative design is an ongoing cycle composed of three steps: design, prototype and evaluate

The Iterative Design Process

1. In the design step – teams consider business goals and customer needs, setting measurable goals and developing design concepts
2. In the prototype step – teams develop artifacts as basic as scenarios and storyboards and as complex as creating running web sites (for example), that illustrate how the site will accomplish these goals.
3. In the evaluate step – teams assess the prototype to see if they meet the desired goals.
4. The results are then used to inform the design in the next iteration, and the entire process repeats until the goals have been met.

Nokia's approach to developing a communicator



Nokia's approach to developing a communicator

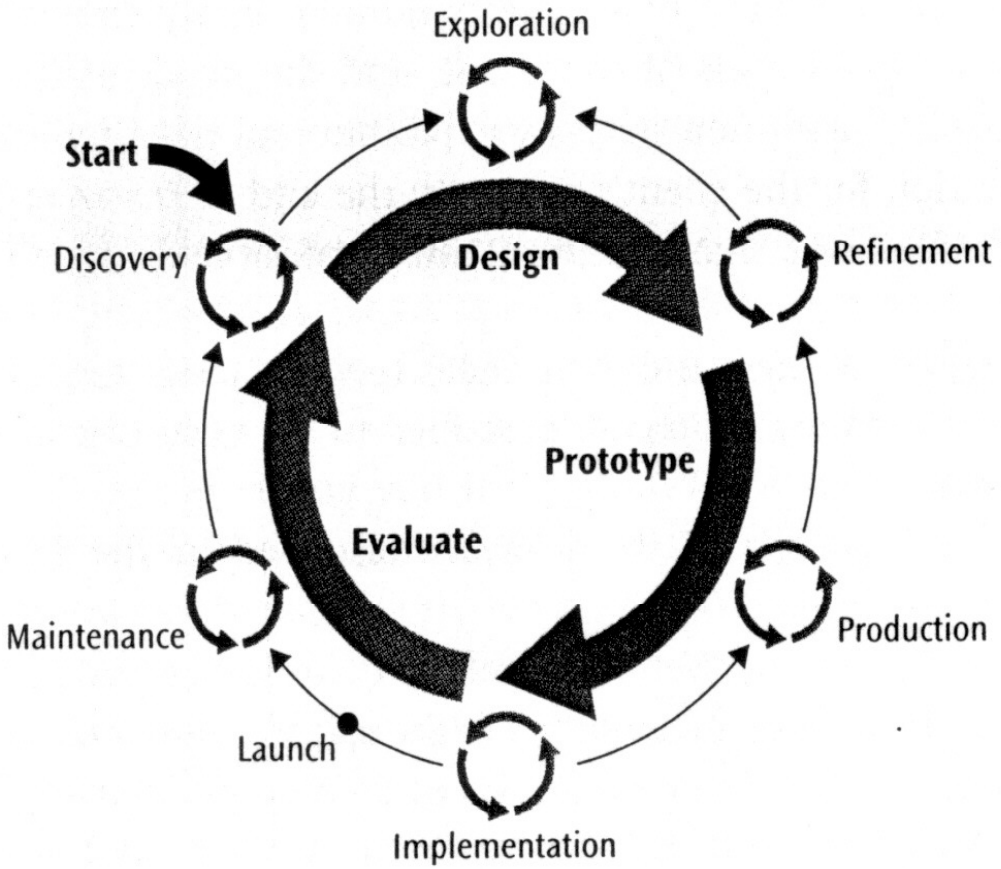
- It has four main steps
- The cycle begins with data gathering. The data is collected through market research studies, data from previous projects and contextual techniques
- Scenarios and then task models are built by analyzing the data collected, and initial designs are proposed

Nokia's approach to developing a communicator

- Many iterations of design and evaluation are performed before the final design emerges.
- During the last iteration phase, the final design is tested with end users and expert usability specialists

Website Development Process

- Both as a whole and at each individual phase, uses iterative design



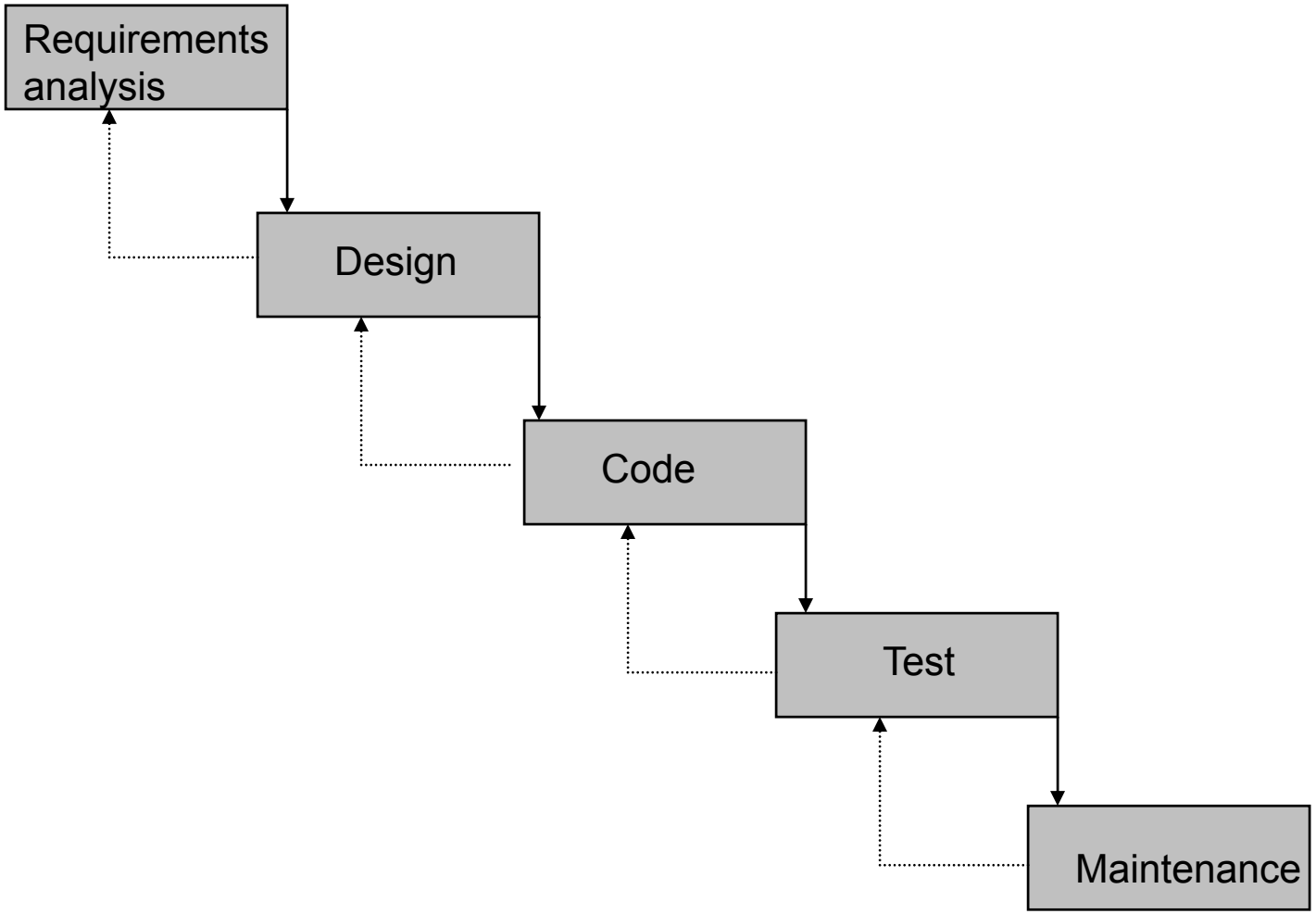
Website Development Process

- Discovery – understanding the target customers and their needs, and conceptualizing the business and customer goals for the web site
- Exploration – generating several rough initial web site designs, of which one or more will be chosen for further development
- Refinement – polishing the navigation, layout and flow of the selected design

Website Development Process

- Production – developing a fully interactive prototype and a design specification
- Implementation – developing the code, content and images for the web site
- Launch – deploying the web site for actual use
- Maintenance – supporting the existing site, gathering and analyzing metrics of success and preparing for the next redesign

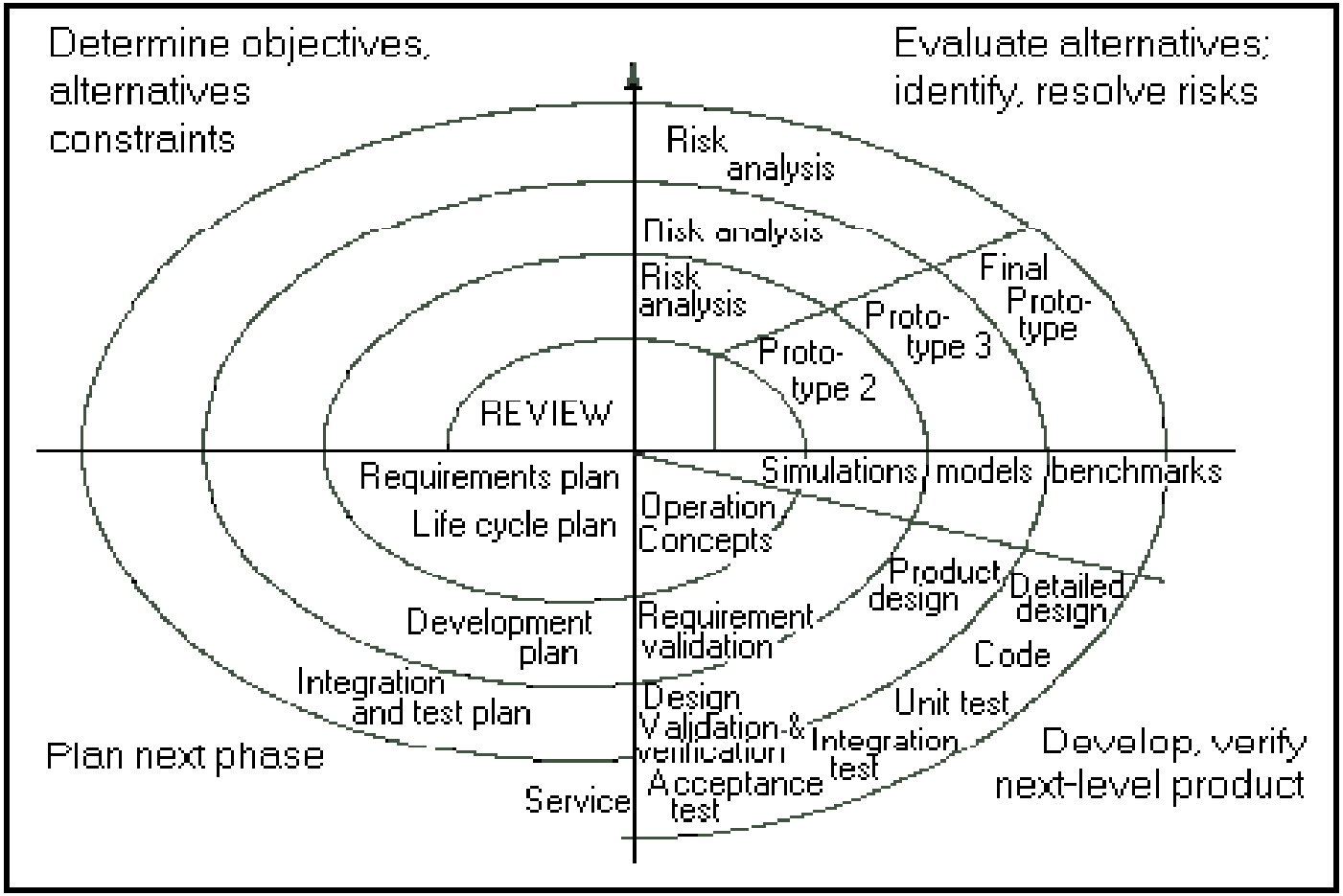
The waterfall lifecycle model



The spiral lifecycle model

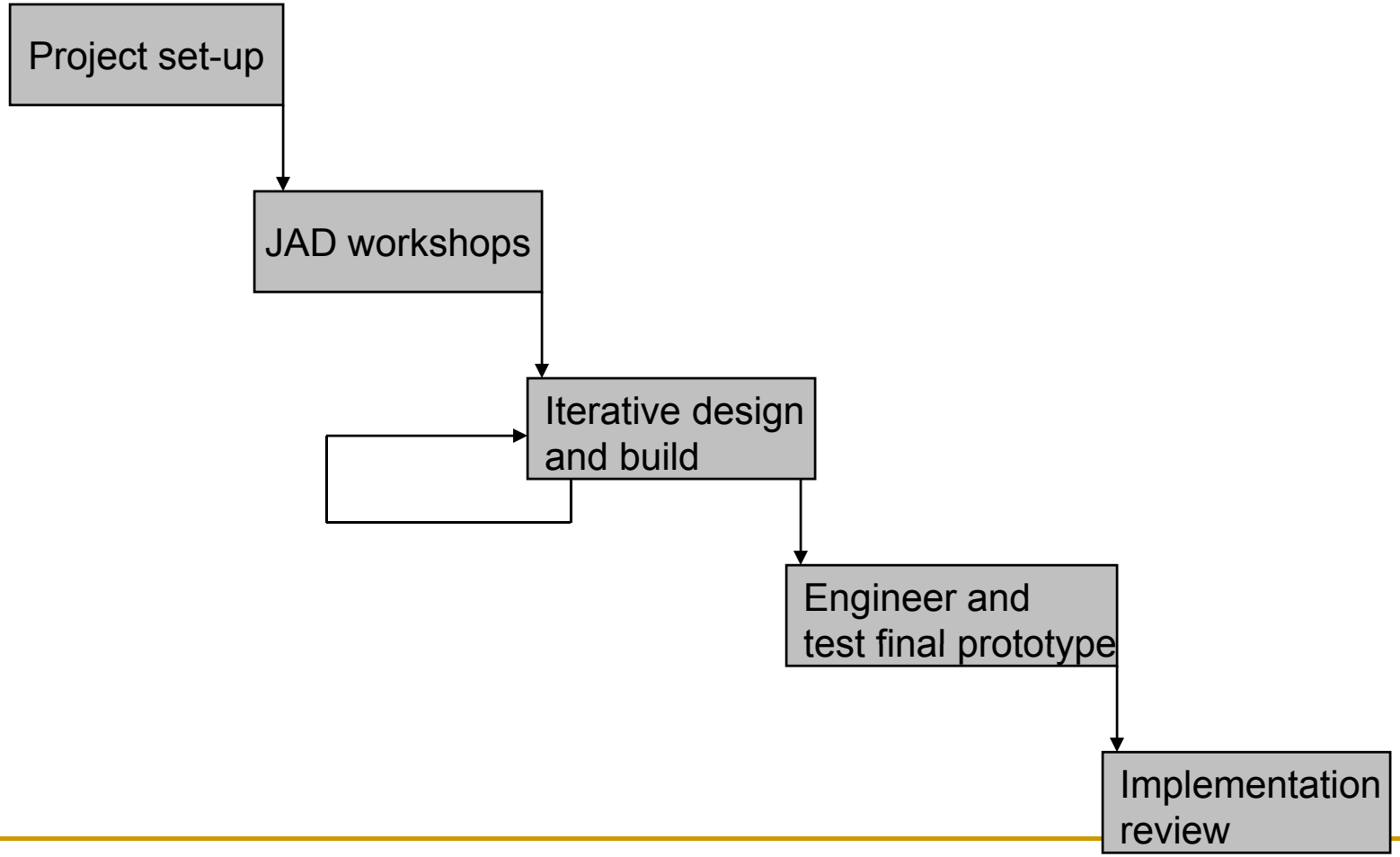
- Important features:
  - Risk analysis
  - Prototyping
  - Iterative framework allows ideas to be checked and evaluated
  - Explicitly encourages alternatives to be considered

The spiral lifecycle model



From cctr.umkc.edu/~kennethjuwng/spiral.htm

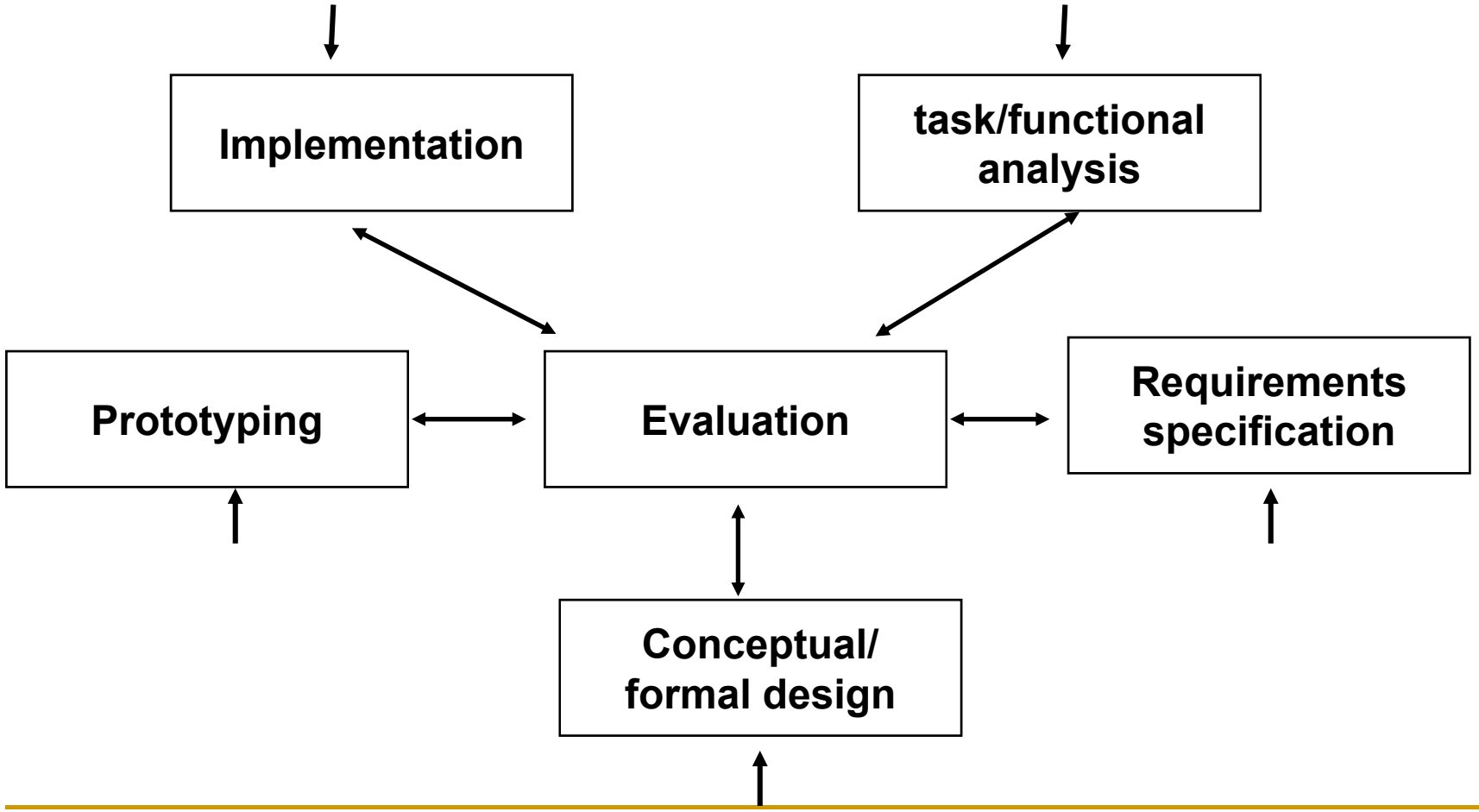
A basic RAD (Rapid Applications Development) lifecycle model



The Star Lifecycle Model

- Important features:
  - Derived from some empirical work of interface designers
  - No particular ordering of activities
  - Evaluation is central to this model

The Star Model



The Usability Engineering Lifecycle Model

- Important features:
  - Holistic (i.e., complete) view of usability engineering
  - Provides links to software engineering approaches
  - Three essential tasks: requirements analysis, design/testing/development, and installation
  - Stages of identifying requirements, designing, evaluating, building prototypes
  - Uses a style guide to capture a set of usability goals
  - Can be scaled down for small projects

Microsoft Development Process

- Attempts to scale up the culture of a loosely-structured, small software team
- Each small team of developers have freedom to evolve their designs and operate nearly autonomously
- All teams synchronize their activities daily and periodically stabilize the whole product, “synch and stabilize”

Planning Phase

- Begins with a vision statement that defines the goals of the new product and supported user activities
- Program managers write functional specifications with enough detail to develop schedules and allocate staff

Development Phase

- Feature list is divided into smaller groups, each with its own small development team
- Schedule is broken up into milestones
- Teams work in parallel and synchronize their work on a daily and weekly basis

Stabilization Phase

- Once a milestone is reached, all errors are found and fixed
- The next milestone is then pursued

Final Products

- Excel, Office, Publisher, Windows 95, Windows NT, Word, and Works, among others were developed with this “synch and stabilize” process

Summary

- Four basic activities in the design process
  - Identifying needs and establishing requirements
  - Developing alternative designs
  - Building interactive versions of the design
  - Evaluating designs
- Three key characteristics of the interaction design process
  - Focus on users
  - Specific usability and user experience goals
  - Iteration
- Lifecycle models show how these are related

Exercise

- Consider the design issues involved for the following:
  - A mobile phone for old people
  - A mobile phone for young children
  - An office phone

Exercise

- Identify the key functionality required
- Identify key user & task characteristics
- Consider the design trade offs that might be required
- State some usability criteria that could be used to evaluate the design
- Produce a ‘front’ end for your device

Design Languages and Implementation Support

Implementation support

- programming tools
  - levels of services for programmers
- windowing systems
  - core support for separate and simultaneous user-system activity
- programming the application and control of dialogue
- interaction toolkits
  - bring programming closer to level of user perception
- user interface management systems
  - controls relationship between presentation and functionality

Introduction

How does HCI affect of the programmer?

Advances in coding have elevated programming

hardware specific

$\rightarrow$ interaction-technique specific

Layers of development tools

- windowing systems
- interaction toolkits
- user interface management systems

Elements of windowing systems

Device independence

programming the abstract terminal device drivers

image models for output and (partially) input

- pixels
- PostScript (MacOS X, NextStep)
- Graphical Kernel System (GKS)
- Programmers' Hierarchical Interface to Graphics (PHIGS)

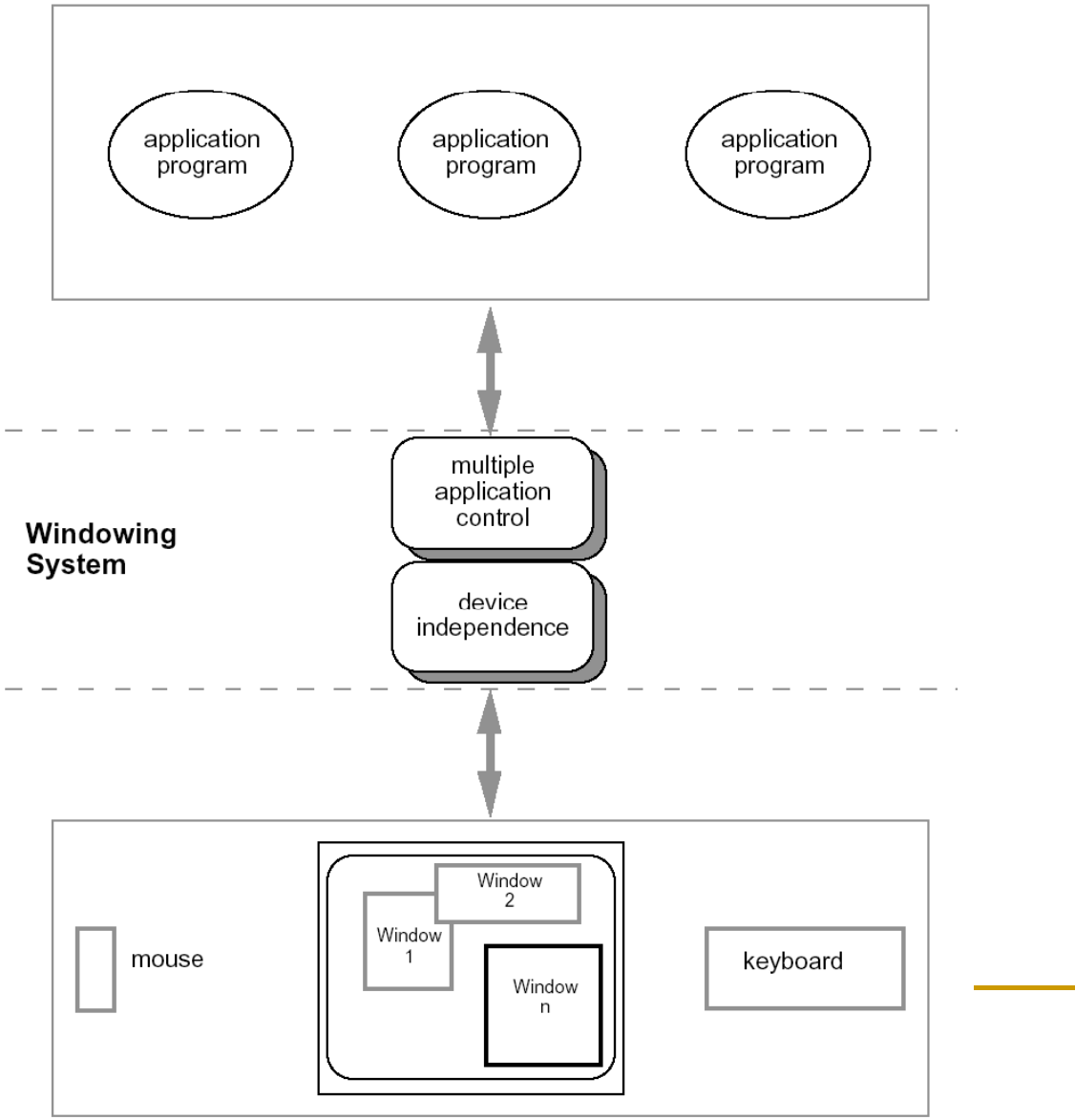
Resource sharing

achieving simultaneity of user tasks

window system supports independent processes

isolation of individual applications

Roles of a windowing system



Architectures of windowing systems

Three possible software architectures

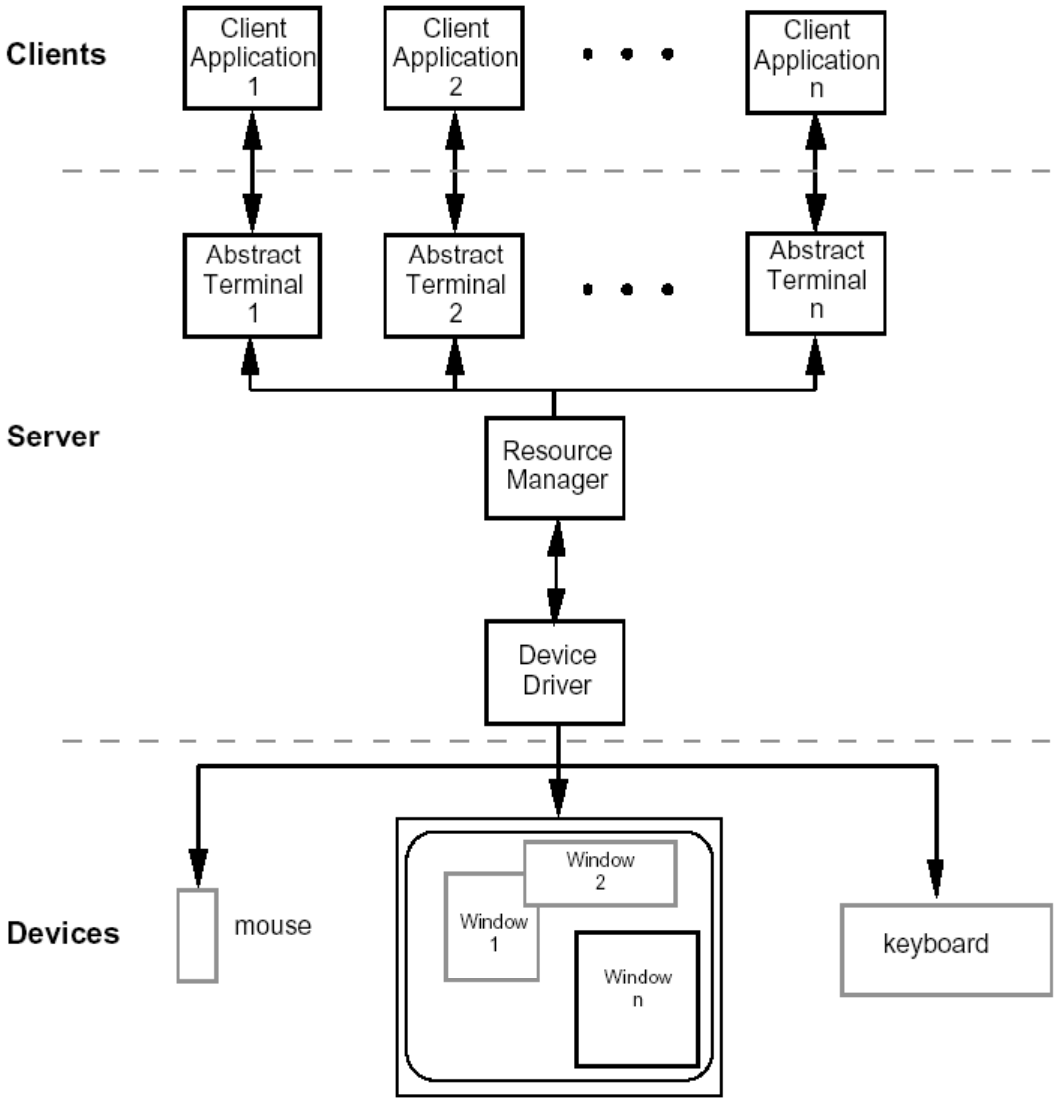
- all assume device driver is separate
- differ in how multiple application management is implemented

1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications

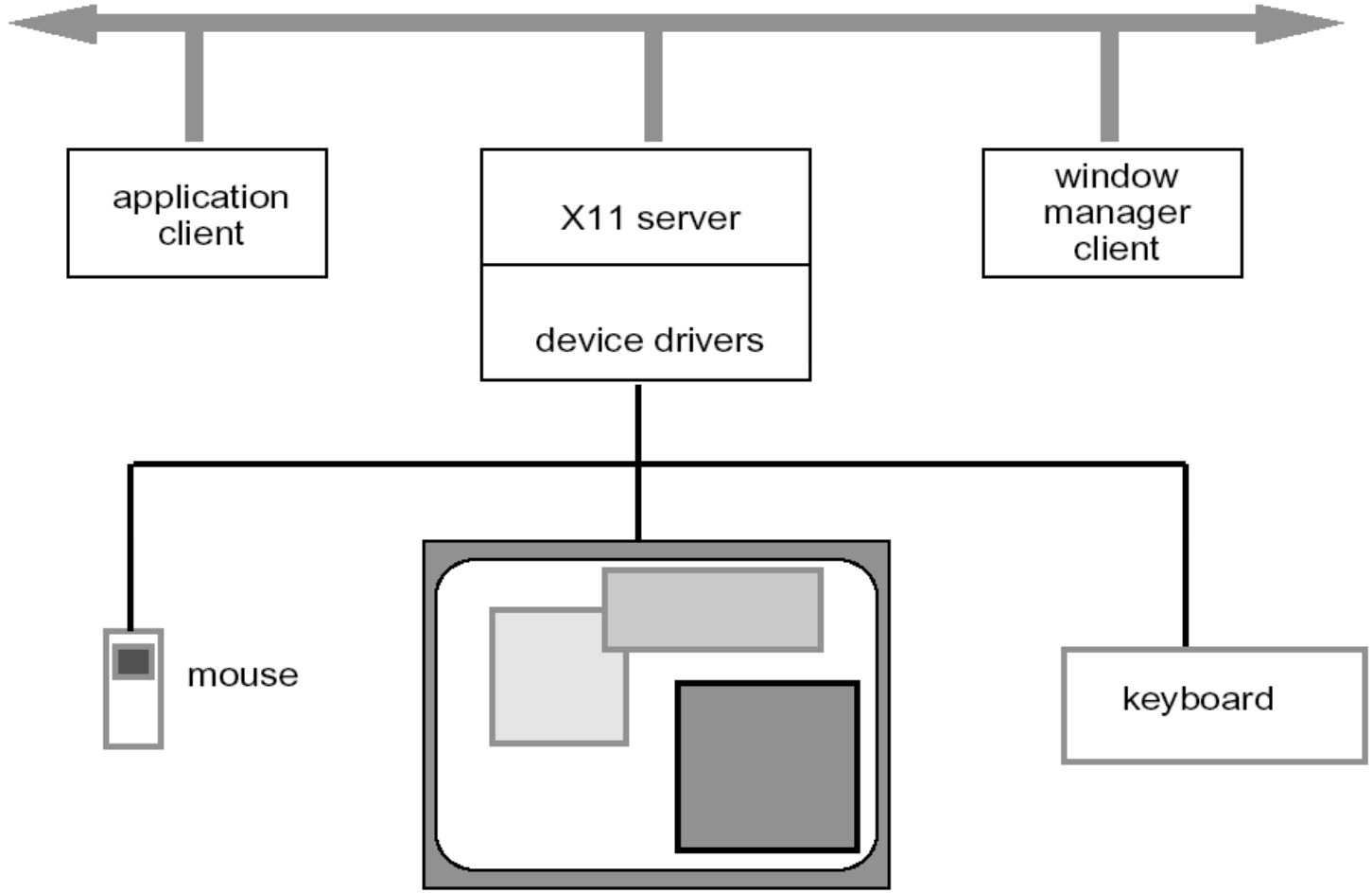
2. management role within kernel of operating system
   - applications tied to operating system

3. management role as separate application
   maximum portability

The client-server architecture



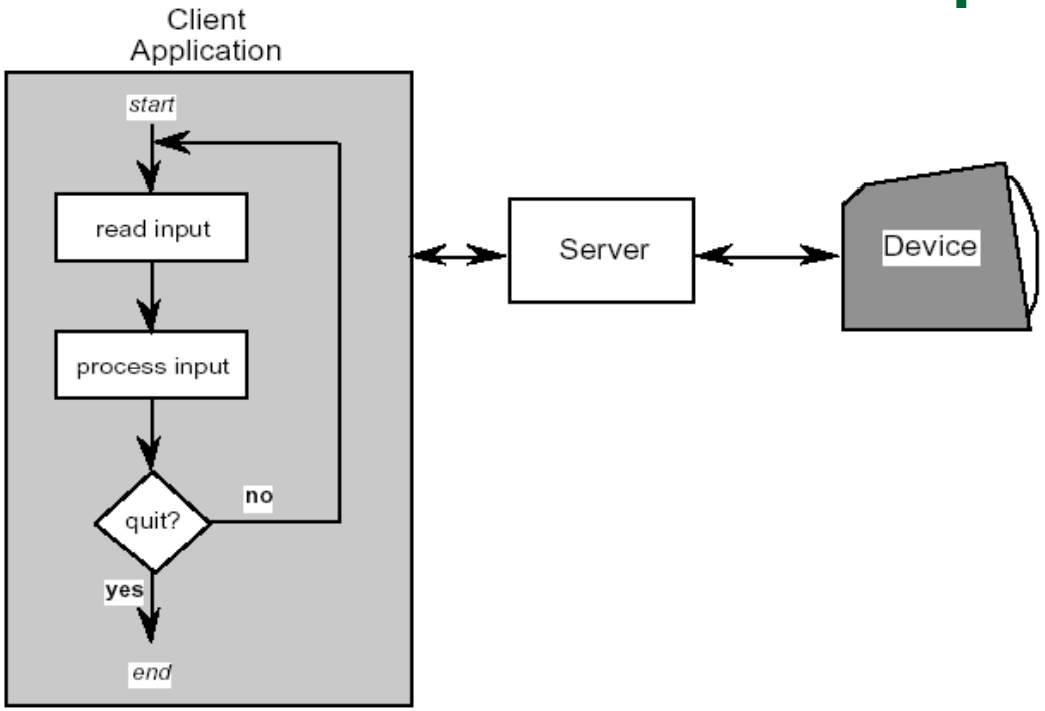
X Windows architecture



X Windows architecture (ctd.)

- pixel imaging model with some pointing mechanism
- X protocol defines server-client communication
- separate window manager client enforces policies for input/output:
  - how to change input focus
  - tiled vs. overlapping windows
  - inter-client data transfer

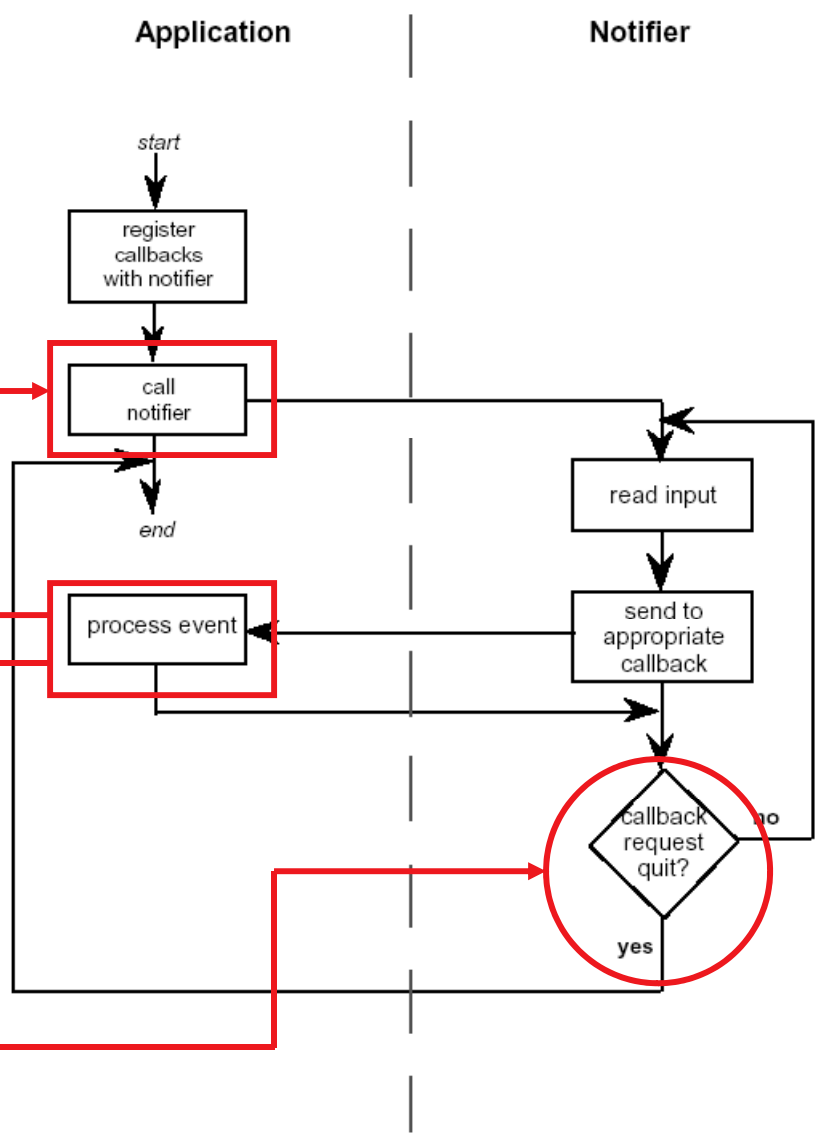
Programming the application - 1
read-evaluation loop




```
repeat
   read-event(myevent)
   case myevent.type
      type_1:
         do type_1 processing
      type_2:
         do type_2 processing
      ...
      type_n:
         do type_n processing
   end case
end repeat
```

Programming the application - 1 notification-based

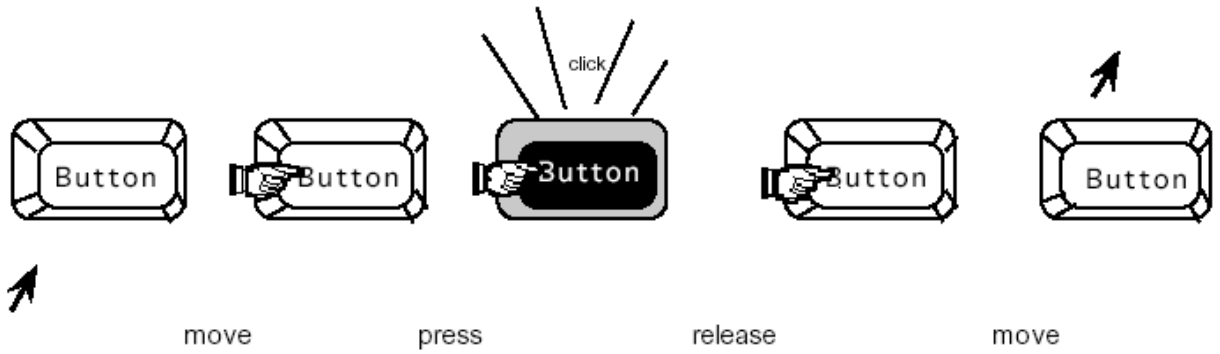
```
void main(String[] args) {
   Menu menu = new Menu();
   menu.setOption("Save");
   menu.setOption("Quit");
   menu.setAction("Save",mySave)
   menu.setAction("Quit",myQuit)
      ...
}

int mySave(Event e) {
   // save the current file
}

int myQuit(Event e) {
   // close down
}
```



Using toolkits

Interaction objects

- input and output intrinsically linked



Toolkits provide this level of abstraction

- programming with interaction objects (or
- techniques, widgets, gadgets)
- promote consistency and generalizability
- through similar look and feel
- amenable to object-oriented programming

Interfaces in Java

- Java toolkit – AWT (abstract windowing toolkit)
- Java classes for buttons, menus, etc.
- Notification based;
  - AWT 1.0 – need to subclass basic widgets
  - AWT 1.1 and beyond -– callback objects
- Swing toolkit
  - built on top of AWT – higher level features
  - Uses MVC (Model - View - Controller) architecture

User Interface Toolkits

- A user interface toolkit is a library of interface objects and related information such as buttons, menu bars, scroll bars, and icons, error messages and help messages
- Provide programmers with ready-made screen items that can be called up by applications as needed
- Early ones were inflexible (couldn't change the items much) but newer ones are based on OO approach (which allows for reusability but also customization of interface objects such as menus)
- Interface builders allow for generation of source code

Integrated Environments

- Problems with using stand-alone tools
  - Transferring data between tools can be difficult and time consuming
  - One or more facilities may be inadequate
  - There may be a lack of consistency among tools used and across user interface styles in different tools
- As a consequence there has been a move towards integrated programming environments for user interface design
  - User interface management systems as one example

Rapid prototyping tools

- Quick development of user experience
- Minimize “programming” knowledge
  - Leverage other metaphors for design
  - GUI builders

Some examples

- HyperCard, SuperCard, Revolution
- HTML and JavaScript
- Visual Basic
- Director and Flash

User Interface Management Systems (UIMSs)

- UIMSs focus on the runtime problems of interface execution
- A UIMS is a high-level interactive software application that facilitates the efficient development of high quality user interfaces
- Mediates the interaction between the end user of an application and the application code itself
- A User Interface Design Environment (UIDE) is an integrated environment that offers facilities for designing the interface, not necessarily for managing the interaction at runtime

UIMS as conceptual architecture

- *separation* between application semantics and presentation
- improves:
  - portability – runs on different systems
  - reusability – components reused cutting costs
  - multiple interfaces – accessing same functionality
  - customizability – by designer and user

User Interface Design Environment (UIDE)

- Allows application programmers and interface designers to create interfaces without programming and without having to learn the details of underlying toolkits
- Through UIDEs, the designer creates an interface for an application by describing the application at the semantic level (in terms of its functionality) using objects and operations
- In practice, a UIDE often offers UIMS facilities (and systems that claim to be UIMSs offer more facilties than simply runtime management)

Example - Garnet

- Garnet is a system described as both a UIMS and a UIDE, developed at Carnegie Mellon University
- Provides an object-oriented approach to user interface design which is intended to ease the development and prototyping of interactive, graphical, direct manipulation user interfaces

Features of Garnet

(1) A custom-built object-oriented programming system that facilitates prototyping

(2) A constraint system that allows the designer to specify constraints on the graphical interface (e.g. that an arrow attached to a box should move when the box is moved)

(3) A graphical object system called Opal, which allows graphical objects to be created and edited

(4) A system for handling input from a variety of devices such as keyboard, mouse, digitizer etc.

(5) A collection of screen items, called gadgets or widgets (Garnet's tool kit)

Garnet – High level design aids

(1) Lapidary, Garnet's interface builder – allows the designer to draw pictures of application specific objects, which the application will create at run-time. These pictures are the prototypes

(2) Jade – the dialogue box creator

(3) Can add programming code through a special interface

Summary

Levels of programming support tools

- Windowing systems
  - device independence
  - multiple tasks
- Paradigms for programming the application
  - read-evaluation loop
  - notification-based
- Toolkits
  - programming interaction objects
- UIMS
  - conceptual architectures for separation
  - techniques for expressing dialogue

Pattern Languages for Interaction Design

- A **pattern language** is a structured method of describing good design practices within a particular domain. It is characterized by
  - Noticing and naming the common problems in a field of interest,
  - Describing the key characteristics of effective solutions for meeting some stated goal,
  - Helping the designer move from problem to problem in a logical way, and
  - Allowing for many different paths through the design process.
- Pattern languages are used to formalize decision-making values whose effectiveness becomes obvious with experience but that are difficult to document and pass on to novices.

Reading assignment #4

- To be posted tonight on the course web page and in the newsgroup