Software Engineering

Lecture 06

Architecture & User Interface Design

Software Architecture

- "The software architecture of a program or computing system is the structure or structures of the system, which comprise the software components, the externally visible properties of those components, and the relationships among them."
- It is not operational software but it is representation that enables software engineer to
  - Analyze the effectiveness of design in meeting its stated requirement.
  - consider architectural alternatives at a stage when making design changes is still relatively easy,
  - reduce the risks associated with the construction of the software.

Importance of Software Architecture

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development
- Architecture highlights early design decisions that will have a profound impact on all software engineering work that follows.
- Architecture “constitutes a relatively small, intellectually graspable model of how the system is structured and how its components work together”

Data Design

- The structure of data has always been an important part of software design.
- *Component level* – the design of *data structures* and the associated algorithms required to manipulate them is essential to the creation of high-quality applications.
- *Application level* – the translation of a data model into a *database design*

Data Design at architectural design

- Today, business (i.e. irrespective of its size large or small) have dozens of database serving many applications encompassing hundreds of gigabytes of data.
- Challenge is to extract useful information from its data environment, particularly when the information desired is cross-functional.
- To solve this challenge, developed technique called
  - Data Mining (also called *knowledge discovery in databases* (KDD)),
  - Data Warehouse
- *Data mining* that navigate through existing databases in an attempt to extract appropriate business-level information

- However, the existence of multiple databases, their different structures, the degree of detail contained with the databases, and many other factors make data mining difficult within an existing database environment.
- A *data warehouse* is a separate data environment that is not directly integrated with day-to-day applications but encompasses all data used by a business
- A data warehouse is a large, independent database that encompasses some, but not all, of the data that are stored in databases that serve the set of applications required by a business.

Data design at the component level

- Component level focuses on the representation of data structures that are directly accessed by one or more software components.

Principles are applicable to data design

1. *The systematic analysis principles applied to function and behavior should also be applied to data.*
2. *All data structures and the operations to be performed on each should be identified.*
3. *A data dictionary should be established and used to define both data and program design (operations)*
4. *Low-level data design decisions should be deferred until late in the design process.*

5. *The representation of data structure should be known only to those modules that must make direct use of the data contained within the structure.*
6. *A library of useful data structures and the operations that may be applied to them should be developed.*
7. *A software design and programming language should support the specification and realization of abstract data types.*

Architectural Style

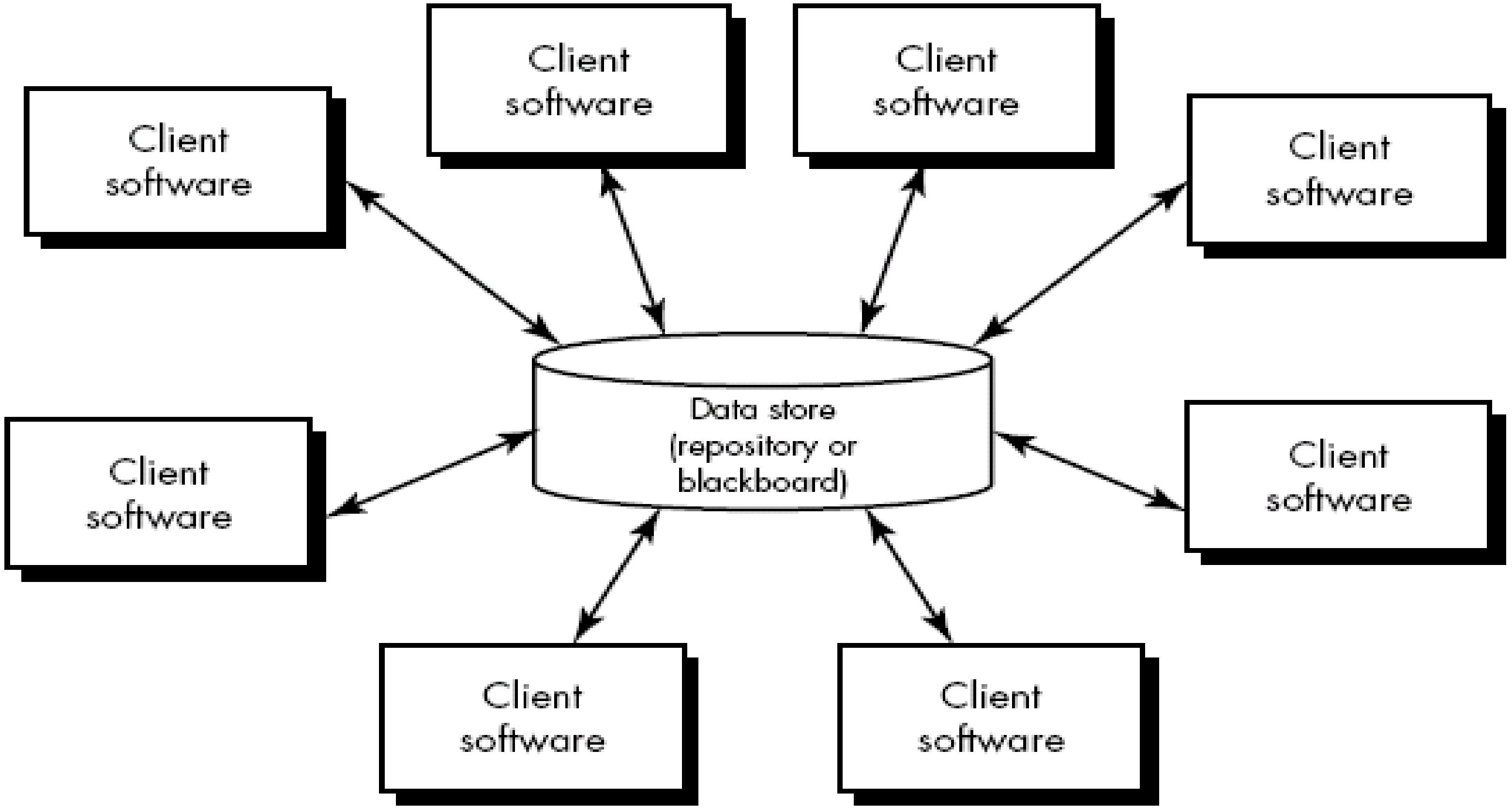
- Style describes a system category that encompasses

1. A set of *components* (e.g., a database, computational modules) that perform a function required by a system;
2. a set of *connectors* that enable "communication, co-ordinations and cooperation" among components;
3. *constraints* that define how components can be integrated to form the system
4. *semantic models* that enable a designer to understand the overall properties of a system

It can be represent by

- Data-centered architecture
- Data flow architecture
- Call and return architecture
- Object oriented architecture
- Layered architecture.

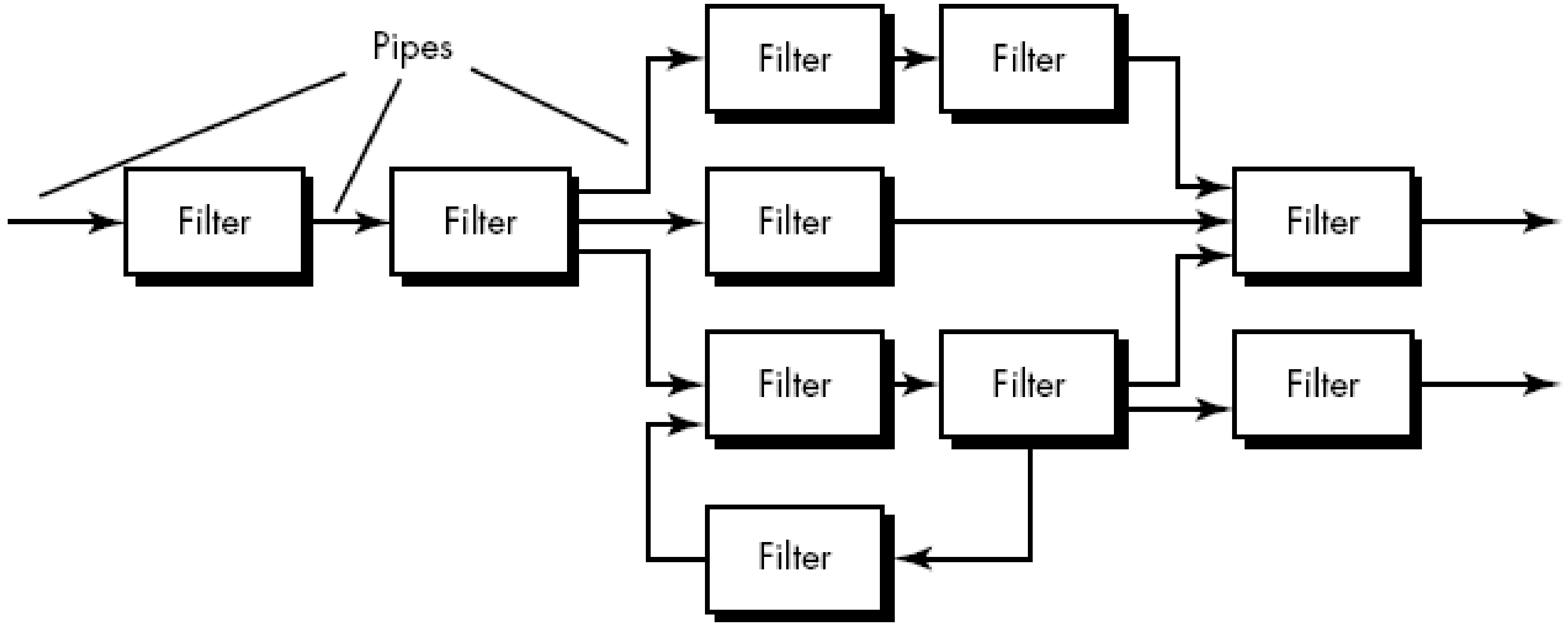
Data-centered architecture



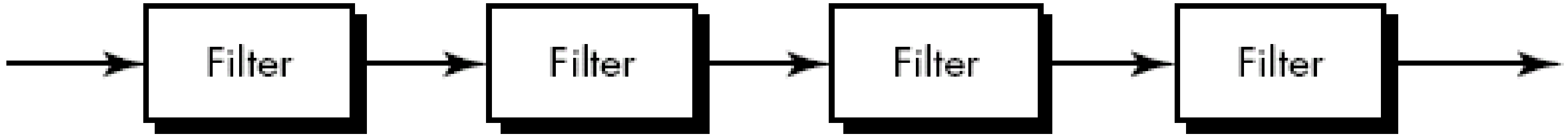
Data-centered architecture

- A data store (e.g., a file or database) resides at the center of this architecture and is accessed frequently by other components that update, add, delete, or otherwise modify data within the store.
- Client software accesses a central repository which is in passive state (in some cases).
- client software accesses the data independent of any changes to the data or the actions of other client software.
- So, in this case transform the repository into a “Blackboard”.
- A blackboard sends notification to subscribers when data of interest changes, and is thus active.
- Data-centered architectures promote *integrability.*
- Existing components can be changed and new client components can be added to the architecture without concern about other clients.
- Data can be passed among clients using the blackboard mechanism. So Client components independently execute processes

Data Flow architecture



Pipes and filters

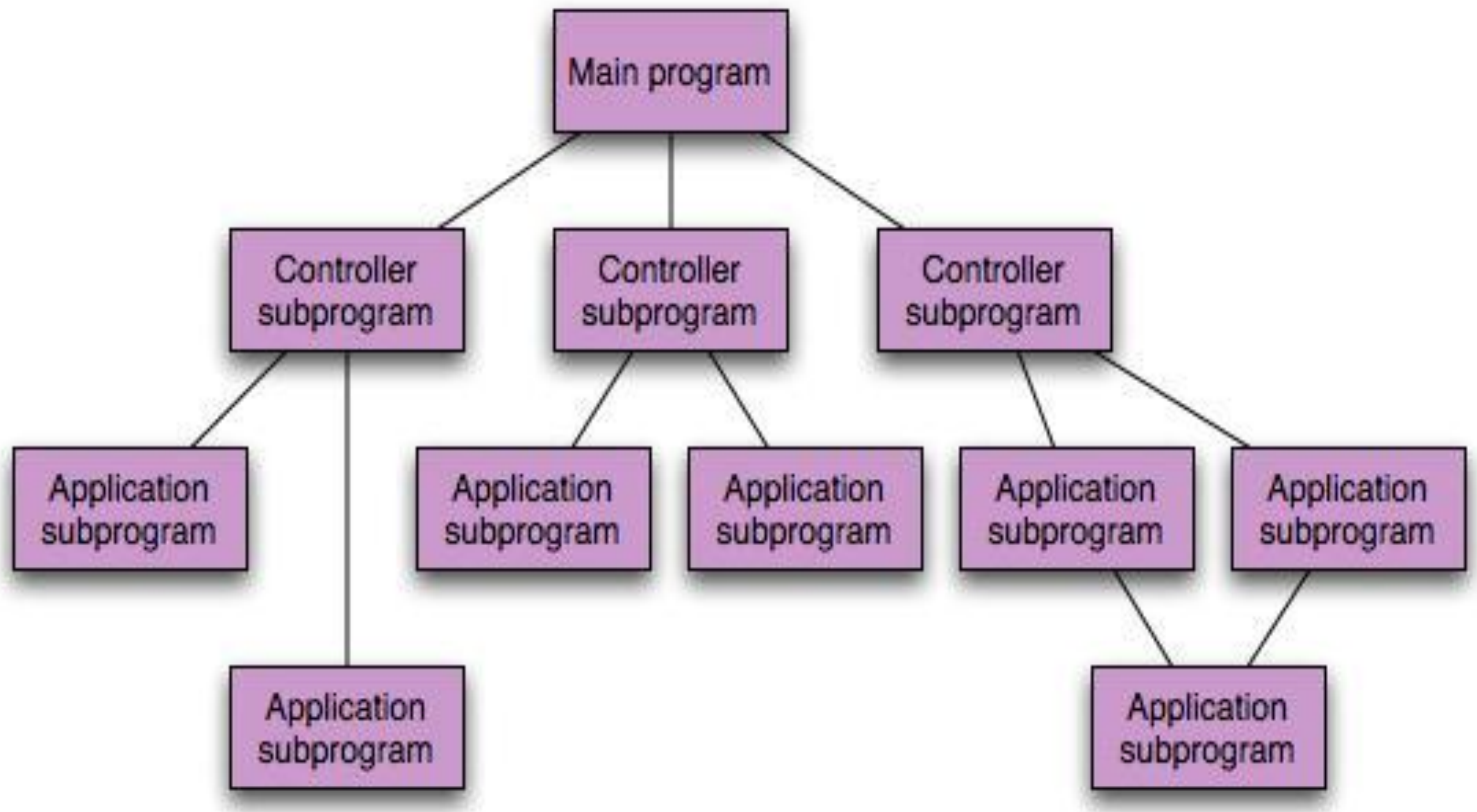


Batch Sequential

Data Flow architecture

- This architecture is applied when input data are to be transformed through a series of computational or manipulative components into output data.
- A *pipe and filter pattern (Fig .1)* has a set of components, called *filters,* connected by pipes that transmit data from one component to the next.
- Each filter works independently (i.e. upstream, downstream) and is designed to expect data input of a certain form, and produces data output (to the next filter) of a specified form.
- the filter does not require knowledge of the working of its neighboring filters.
- If the data flow degenerates into a single line of transforms, it is termed *batch sequential.*

Call and return architecture



Call and return architecture

- Architecture style enables a software designer (system architect) to achieve a program structure that is relatively easy to modify and scale.
- Two sub-styles exist within this category:

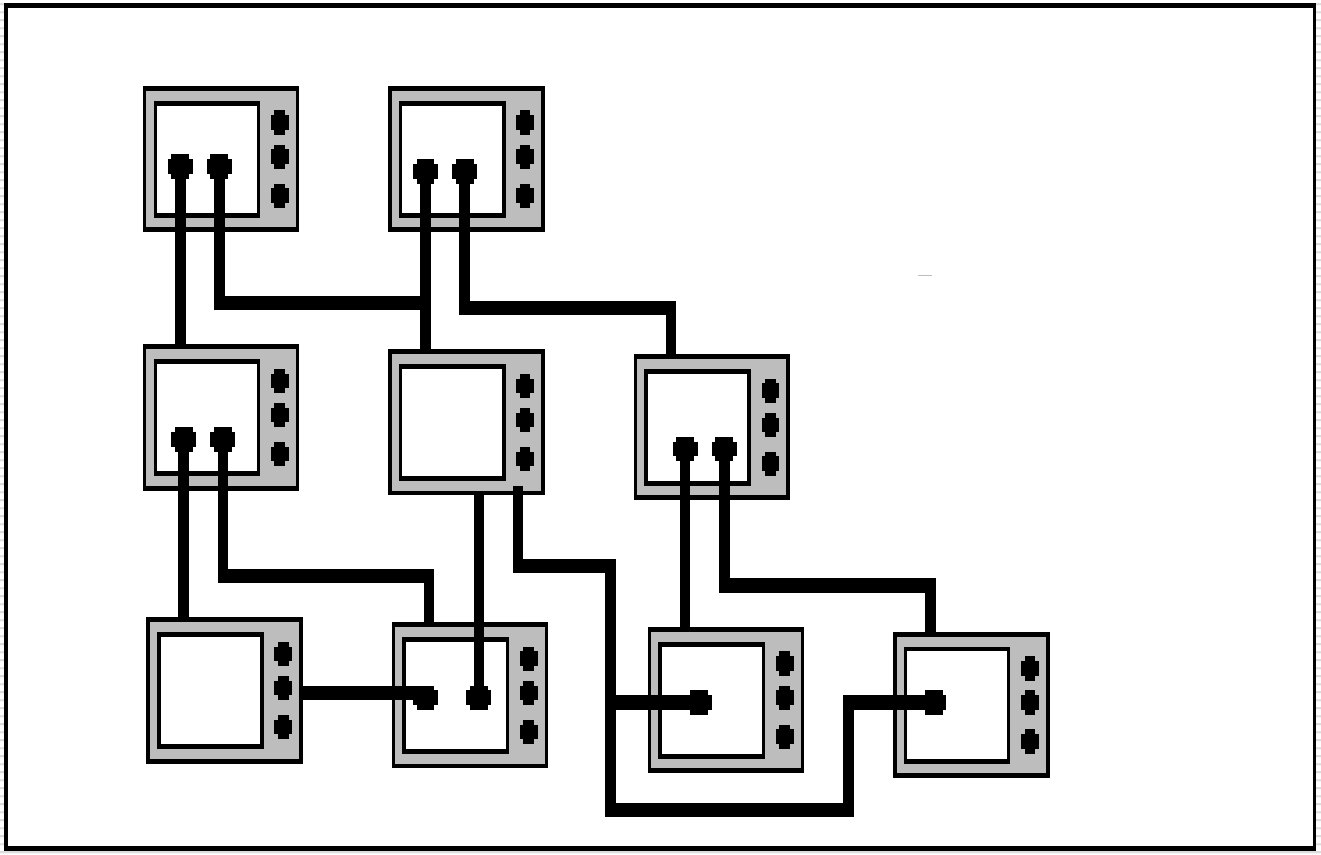
1. *Main/sub program architecture:*

- Program structure decomposes function into a control hierarchy where a “main” program invokes a number of program components, which in turn may invoke still other components.

2. *Remote procedure Call architecture:*

- The components of a main program/subprogram architecture are distributed across multiple computers on a network

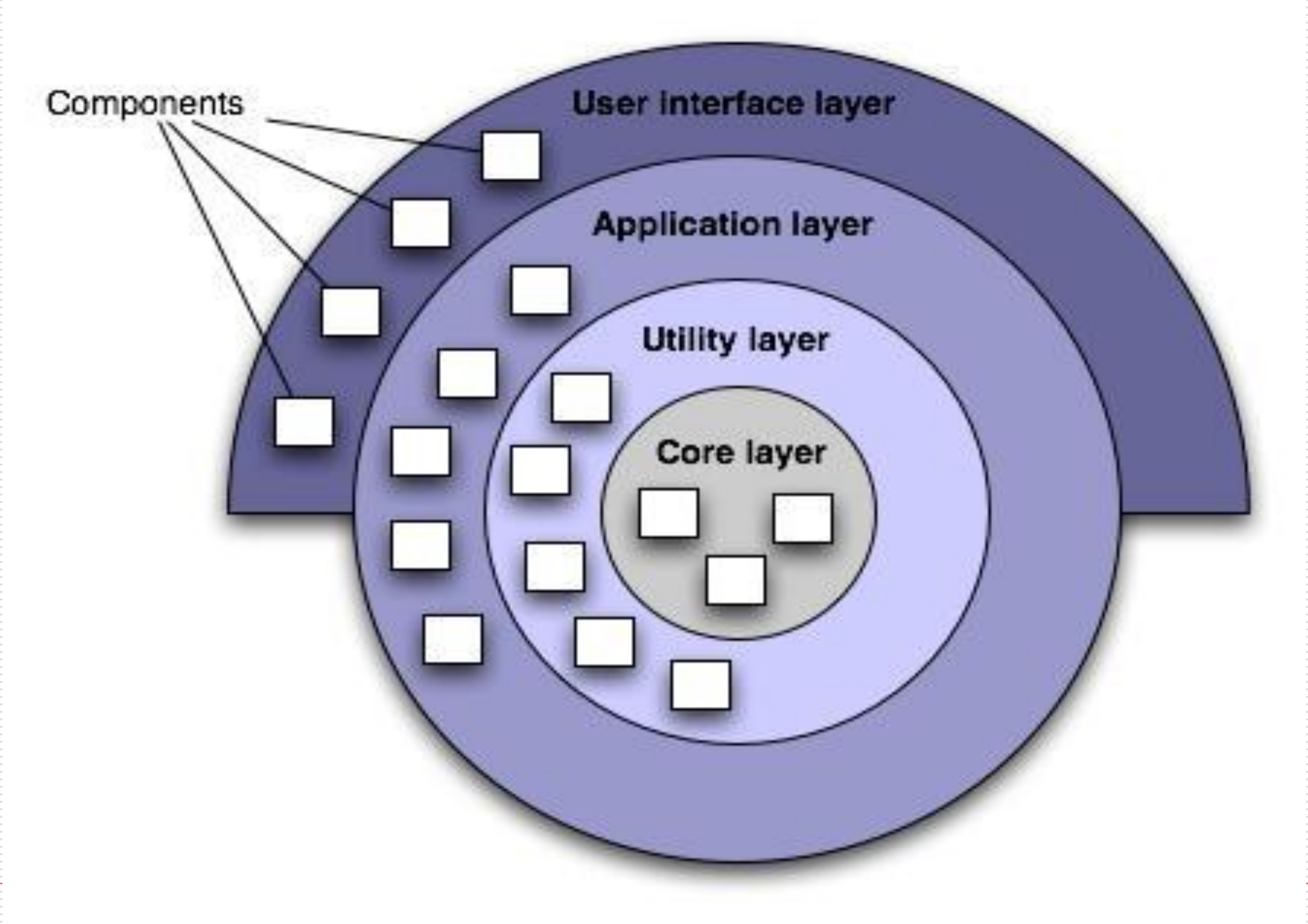
Object-oriented architecture



Object-oriented architecture

- The object-oriented paradigm, like the abstract data type paradigm from which it evolved, emphasizes the bundling of data and methods to manipulate and access that data (Public Interface).
- Components of a system summarize data and the operations that must be applied to manipulate the data.
- Communication and coordination between components is accomplished via message passing.

Layered Architecture

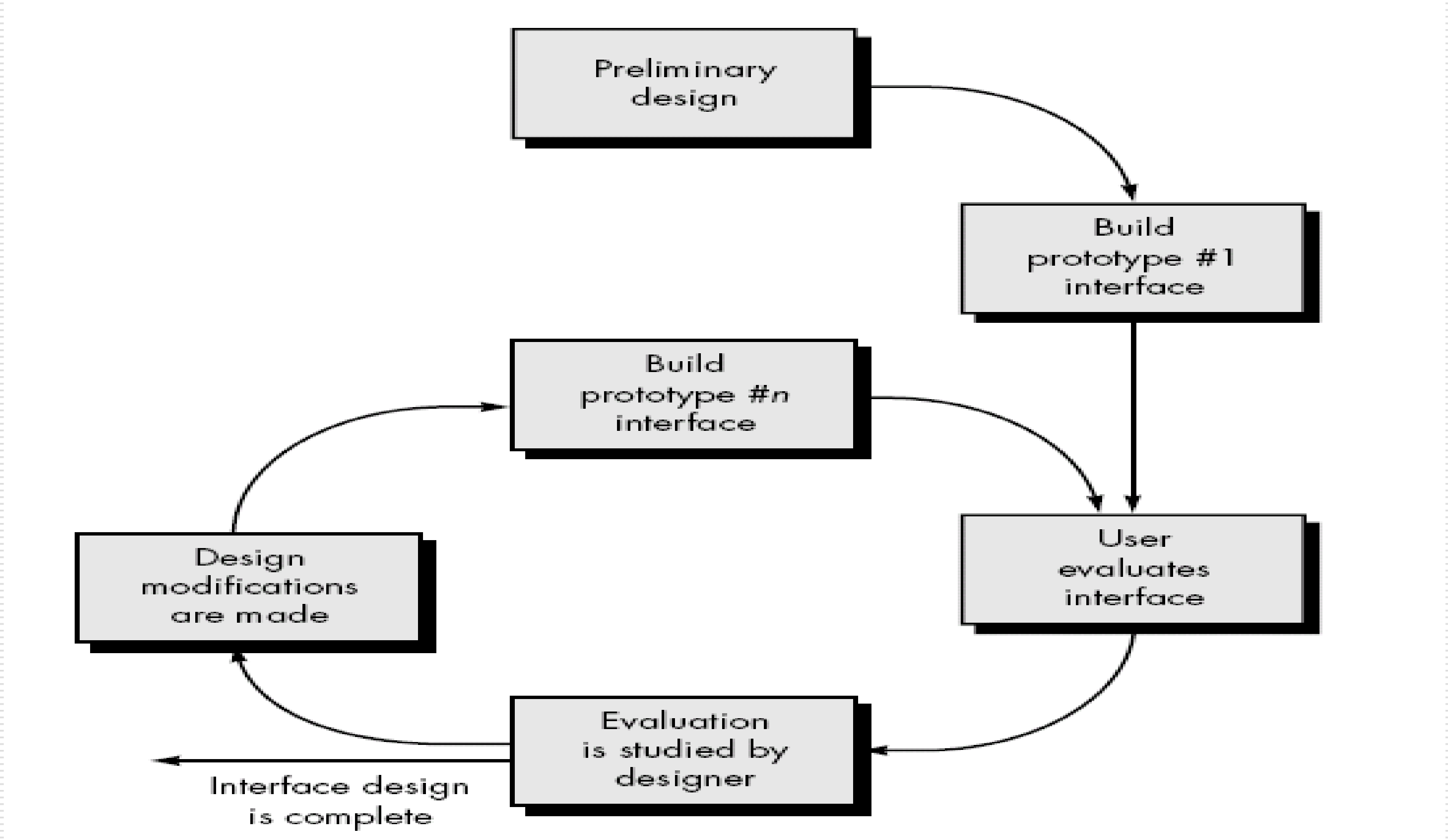


- A number of different layers are defined, each accomplishing operations that progressively become closer to the machine instruction set.
- At the outer layer, components examine user interface operations.
- At the inner layer, components examine operating system interfacing.
- Intermediate layers provide utility services and application software functions.

User Interface Design

- User interface design creates an effective communication medium between a human and a computer.
- Following a set of interface design principles, design identifies interface objects and actions and then creates a screen layout that forms the basis for a user interface prototype.

Design Evaluation


Preliminary design
Build prototype #1 interface
Build prototype #n interface
User evaluates interface
Design modifications are made
Evaluation is studied by designer
Interface design is complete

user interface evaluation cycle

- After the design model has been completed, a first-level prototype is created.
- The prototype is evaluated by the user, who provides the designer with direct comments about the efficiency of the interface.
- In addition, if formal evaluation techniques are used (e.g., questionnaires, rating sheets), the designer may extract information from these data.
- Design modifications are made based on user input and the next level prototype is created.

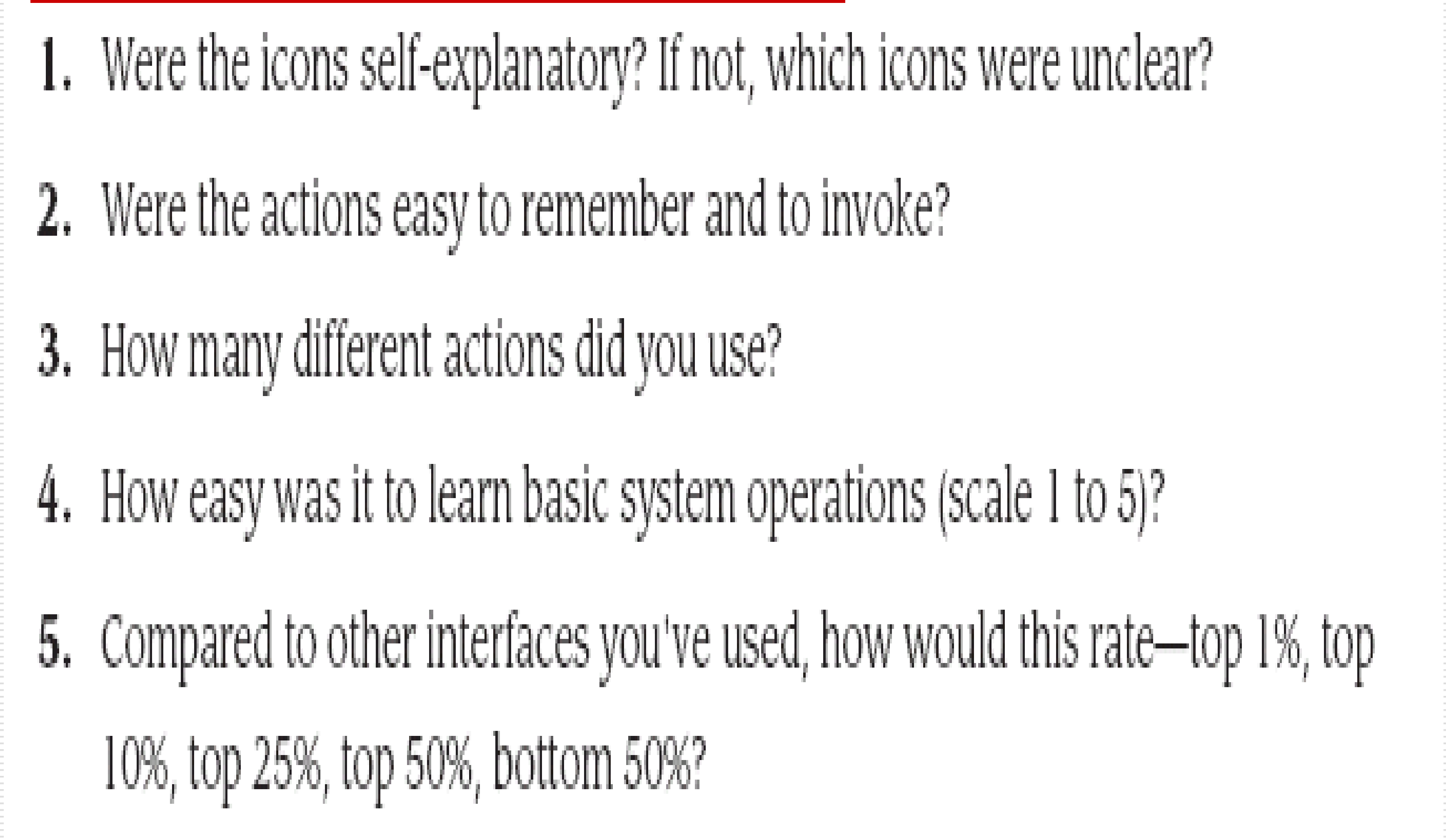
- The evaluation cycle continues until no further modifications to the interface design are necessary.
- The prototyping approach is effective, but is it possible to evaluate the quality of a user interface before a prototype is built?
- If potential problems uncovered and corrected early, the number of loops through the evaluation cycle will be reduced and development time will shorten.

- Evaluation criteria can be applied during early design reviews:

1. The length and complexity of the written specification of the system and its interface provide an indication of the amount of learning required by users of the system.
2. The number of user tasks specified and the average number of actions per task provide an indication of interaction time and the overall efficiency of the system.
3. The number of actions, tasks, and system states indicated by the design model imply the memory load on users of the system.
4. Interface style, help facilities, and error handling protocol provide a general indication of the complexity of the interface and the degree to which it will be accepted by the user.

- Once the first prototype is built, the designer can collect a variety of qualitative and quantitative data that will assist in evaluating the interface.
- To collect qualitative data, questionnaires can be distributed to users of the prototype.
- Questions can be all
  - simple yes/no response,
  - numeric response,
  - scaled (subjective) response,
  - percentage (subjective) response.
  - Likert scale (e.g. strongly disagree, somewhat agree).
  - Open-minded.

Example



1. Were the icons self-explanatory? If not, which icons were unclear?
2. Were the actions easy to remember and to invoke?
3. How many different actions did you use?
4. How easy was it to learn basic system operations (scale 1 to 5)?
5. Compared to other interfaces you've used, how would this rate—top 1%, top 10%, top 25%, top 50%, bottom 50%?

- If quantitative data desired, a form of time study analysis can be conducted.
- Users are observed during interaction, and data such as
  - number of tasks correctly completed over a standard time period,
  - frequency & sequence of actions,
  - time spent "looking" at the display,
  - number and types of errors,
  - error recovery time,
  - time spent using help, and number of help references per standard time period.