SWE 530: Software Design Process

Transferring Design Knowledge

Dr. H. Birkan YILMAZ
Department of Computer Engineering
Boğaziçi University

(birkan.yilmaz@bogazici.edu.tr)

Adapted from slides of Dr. Albert Ali Salah & Başak Aydemir

Summary

- The need to share knowledge
- The architecture concept
- Design methods
- Design patterns
- A unified interpretation

Sharing knowledge on computing

- Three kinds of results (Brooks, 1988):
  - Findings
  - Observations
  - Rules of thumb
- Factors that constrain transfer
  - Invisibility of the medium
  - Influence of implementation
  - Domain factors*
  - Process versus product

Denver Baggage Handling System

http://calleam.com/WTPF/wp-content/uploads/articles/DIABaggage.pdf

-Airport sat idle for 16 months
-560 M USD increase in the overall costs

Mechanisms of design knowledge transfer

- Case studies:
  - specific cases
- Software architecture:
  - seeks to classify the more abstract features of a design solution
- Software design method:
  - a procedural form of guidance on how to develop solutions for specific classes of problems
- Design patterns
  - solution strategies

Software architecture

- Architectural style:
  - a very **abstract description** of a particular set of **general characteristics** of a solution
  - ‘defines constraints on the form and structure of a family of architectural instances’ (Garlan and Perry, 1995)
  - tells us something about the type of **constructional notations** that are likely to be useful for **describing a particular solution**
  - e.g. client-server, pipeline, classical objects

Impact of architecture

- Understanding
- Reuse
- Evolution
- Analysis
- Management

Classifying architectural style

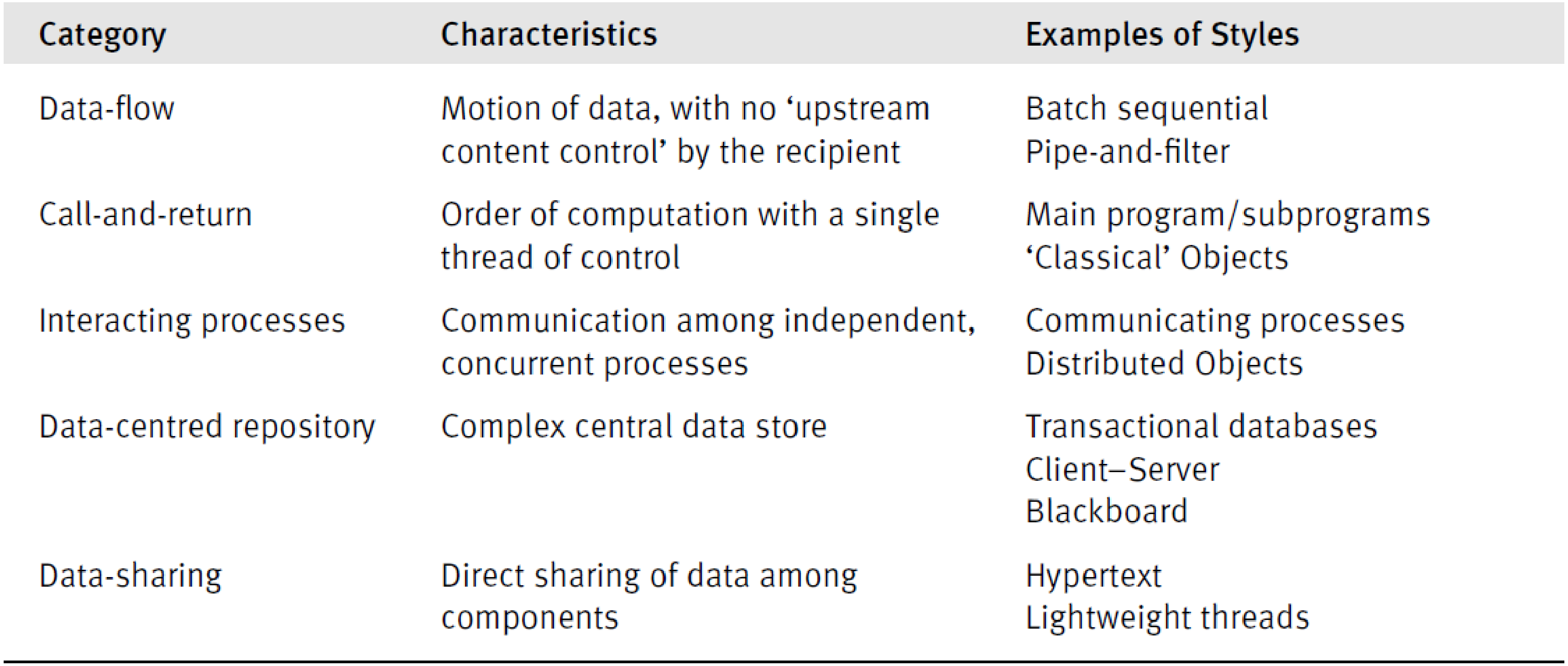
- *Software Architecture = {Elements, Form, Rationale}*

- Elements
  - Entities, processing, data, and connections
- Form
  - ‘detailed properties and relationships’
- Rationale
  - to capture the motivation for particular choices

Classifying architectural style

- the kinds of components and connectors that are used in the style;
- the ways in which control (of execution) is shared, allocated and transferred among the components;
- how data is communicated through the system;
- how data and control interact;
- the type of (design) reasoning that is compatible with the style.

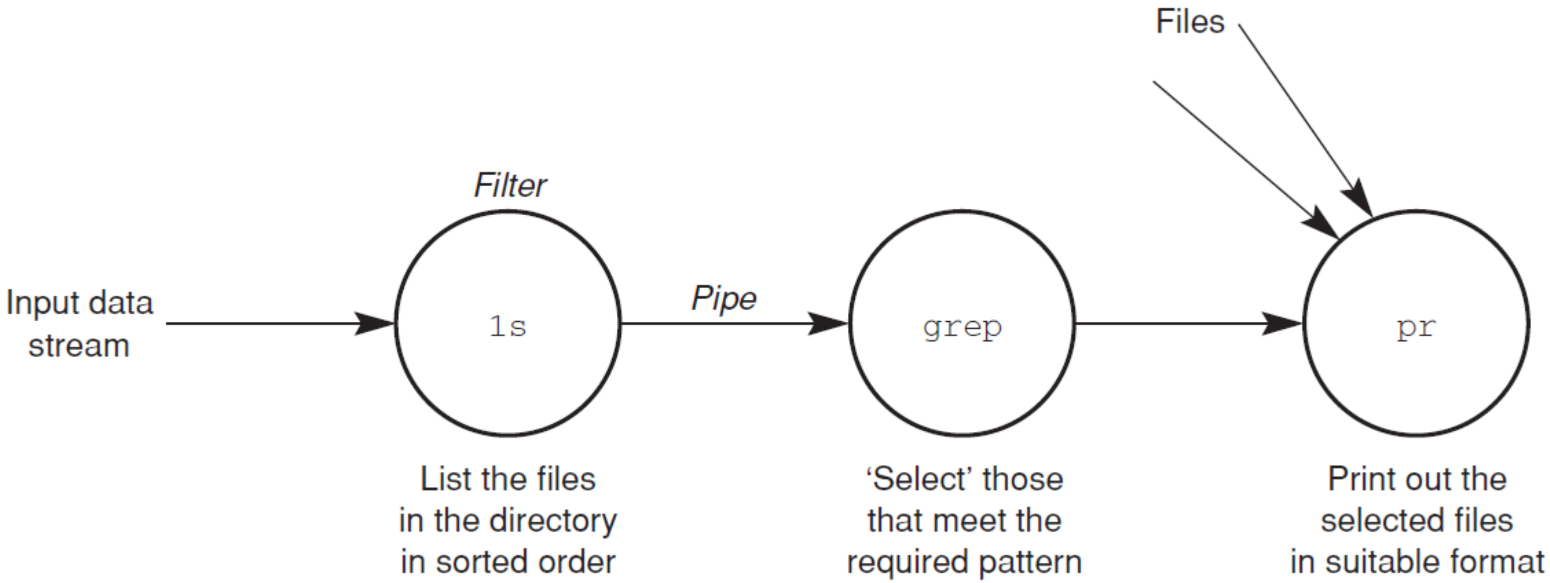
Major categories of arch. style



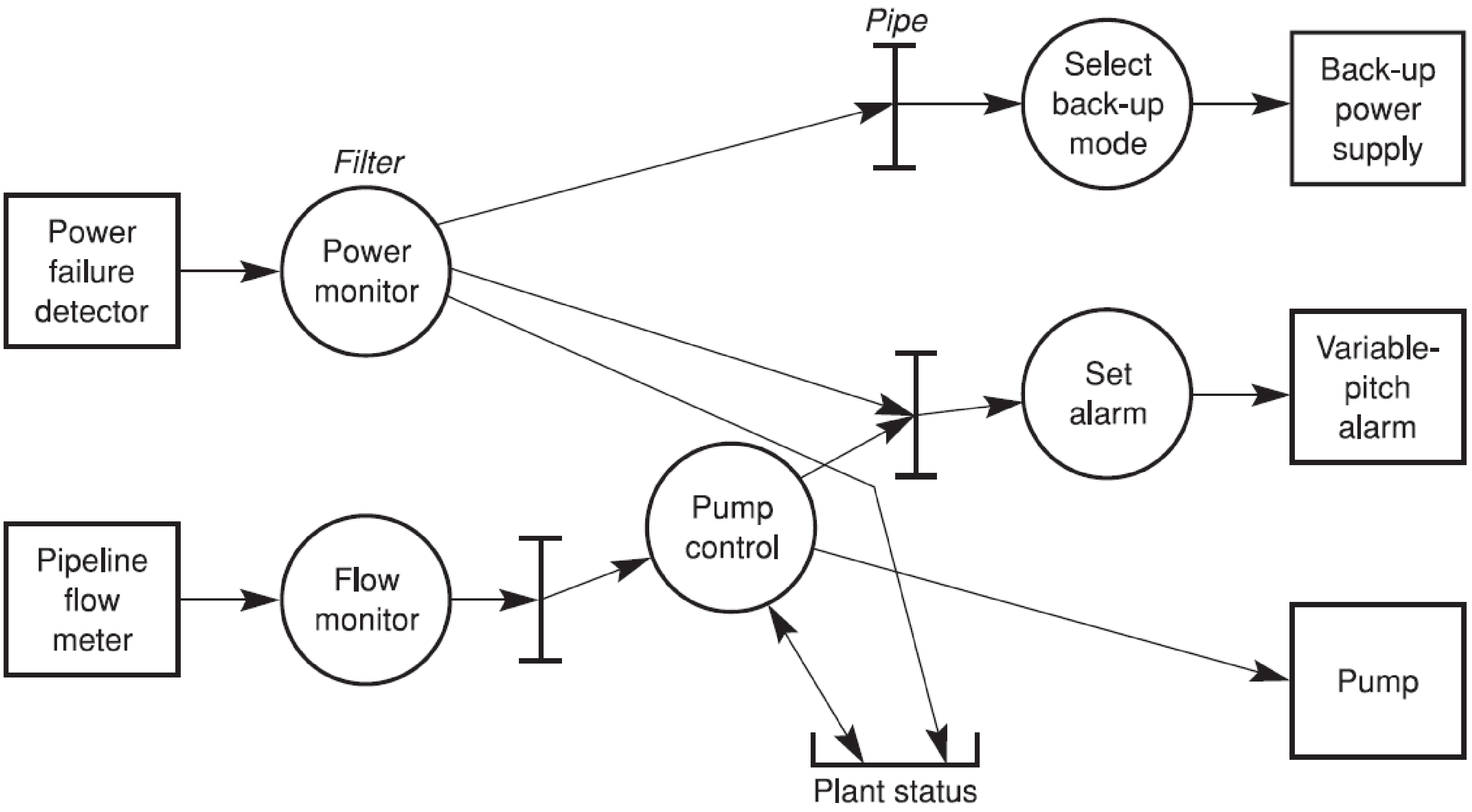
Category	Characteristics	Examples of Styles
Data-flow	Motion of data, with no 'upstream content control' by the recipient	Batch sequential Pipe-and-filter
Call-and-return	Order of computation with a single thread of control	Main program/subprograms 'Classical' Objects
Interacting processes	Communication among independent, concurrent processes	Communicating processes Distributed Objects
Data-centred repository	Complex central data store	Transactional databases Client–Server Blackboard
Data-sharing	Direct sharing of data among components	Hypertext Lightweight threads

Example: Pipe and filter

- Unix uses this style extensively since 70s
- Centred around the **dataflow** that occurs within some form of network

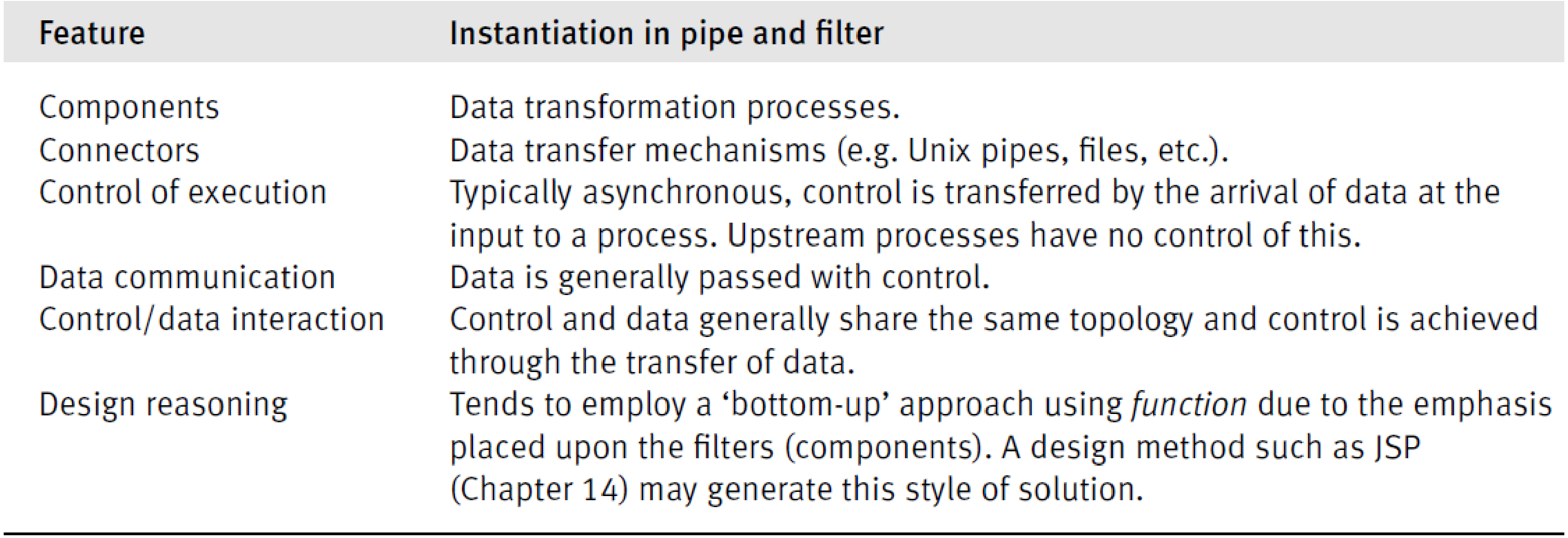

Files
Filter
Input data stream
Pipe
ls
grep
pr
List the files in the directory in sorted order
'Select' those that meet the required pattern
Print out the selected files in suitable format

Example: Pipe and filter – MASCOT diagram



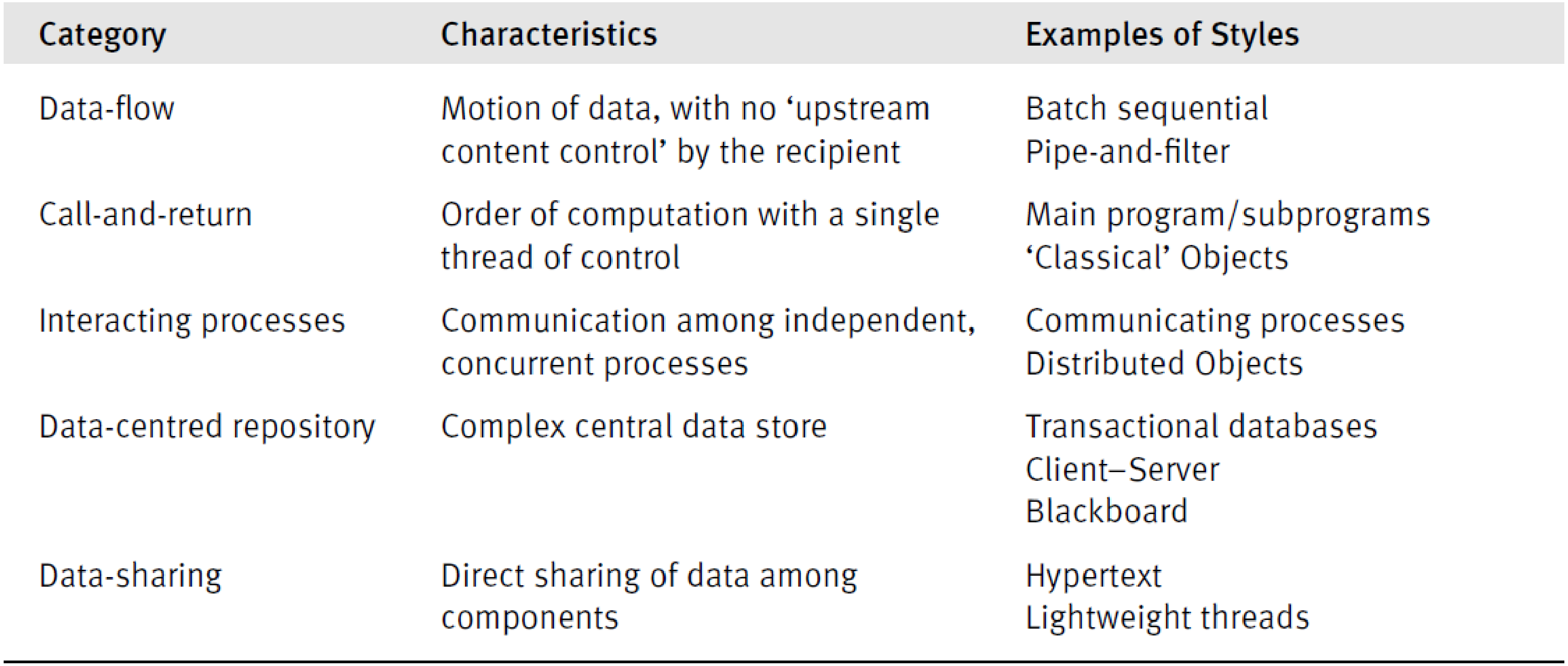
MASCOT: Modular Approach to Software Construction, Operation and Test

Example: Pipe and filter - Features



Feature	Instantiation in pipe and filter
Components	Data transformation processes.
Connectors	Data transfer mechanisms (e.g. Unix pipes, files, etc.).
Control of execution	Typically asynchronous, control is transferred by the arrival of data at the input to a process. Upstream processes have no control of this.
Data communication	Data is generally passed with control.
Control/data interaction	Control and data generally share the same topology and control is achieved through the transfer of data.
Design reasoning	Tends to employ a 'bottom-up' approach using *function* due to the emphasis placed upon the filters (components). A design method such as JSP (Chapter 14) may generate this style of solution.

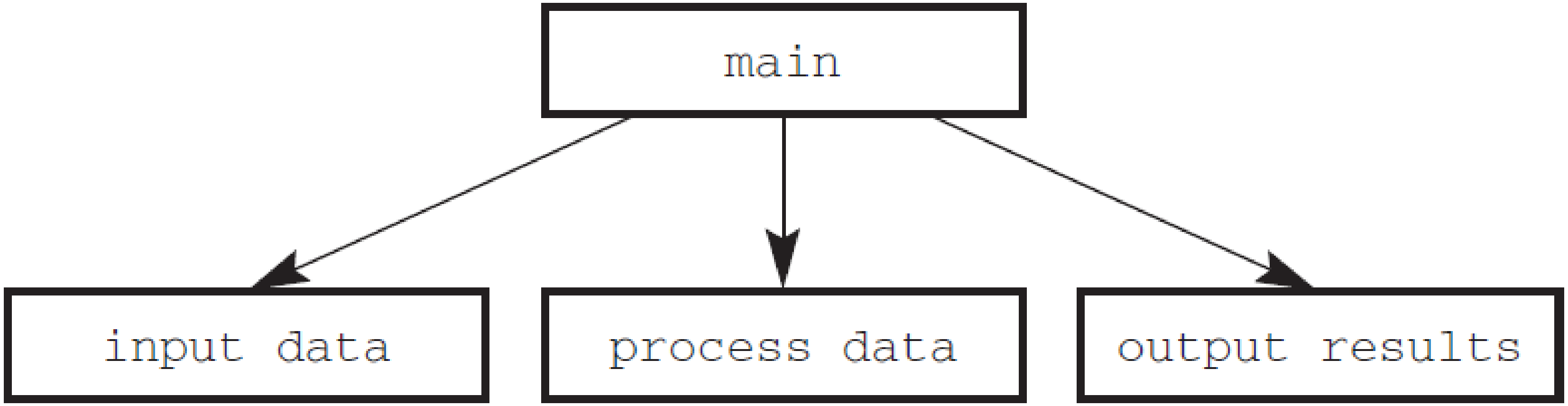
Major categories of arch. style



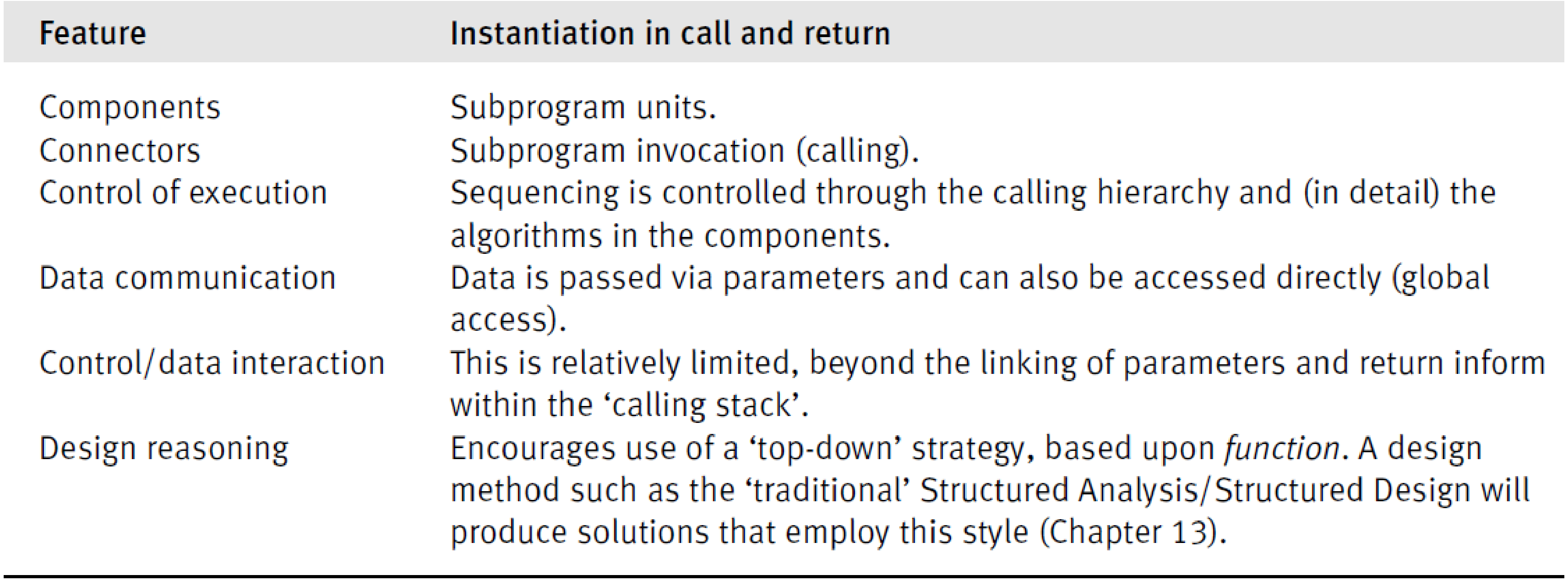
Category	Characteristics	Examples of Styles
Data-flow	Motion of data, with no ‘upstream content control’ by the recipient	Batch sequential Pipe-and-filter
Call-and-return	Order of computation with a single thread of control	Main program/subprograms ‘Classical’ Objects
Interacting processes	Communication among independent, concurrent processes	Communicating processes Distributed Objects
Data-centred repository	Complex central data store	Transactional databases Client–Server Blackboard
Data-sharing	Direct sharing of data among components	Hypertext Lightweight threads

Example: Call and return

- Ordered and hierarchical transfer of control from one processing element to another
- Greater emphasis on control

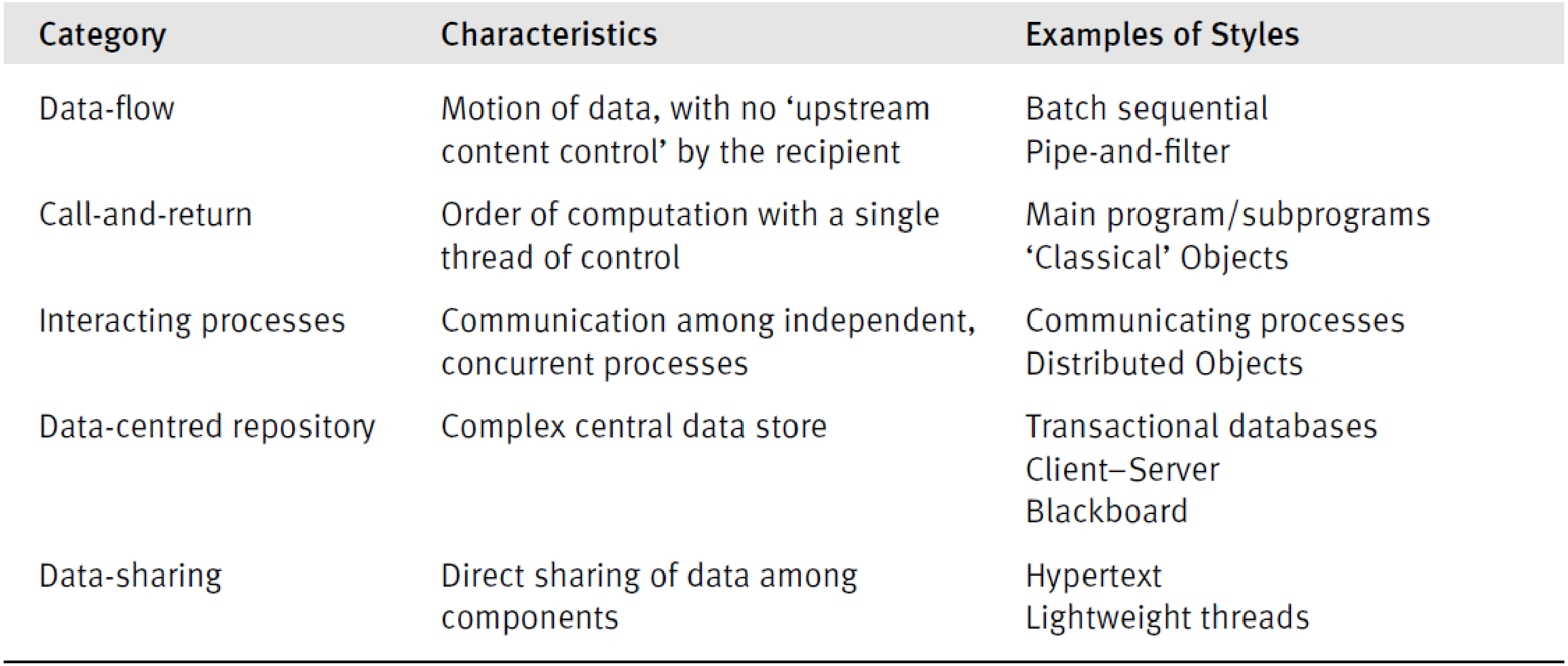


Example: Call and return - Features



Feature	Instantiation in call and return
Components	Subprogram units.
Connectors	Subprogram invocation (calling).
Control of execution	Sequencing is controlled through the calling hierarchy and (in detail) the algorithms in the components.
Data communication	Data is passed via parameters and can also be accessed directly (global access).
Control/data interaction	This is relatively limited, beyond the linking of parameters and return inform within the 'calling stack'.
Design reasoning	Encourages use of a 'top-down' strategy, based upon *function*. A design method such as the 'traditional' Structured Analysis/Structured Design will produce solutions that employ this style (Chapter 13).

Major categories of arch. style

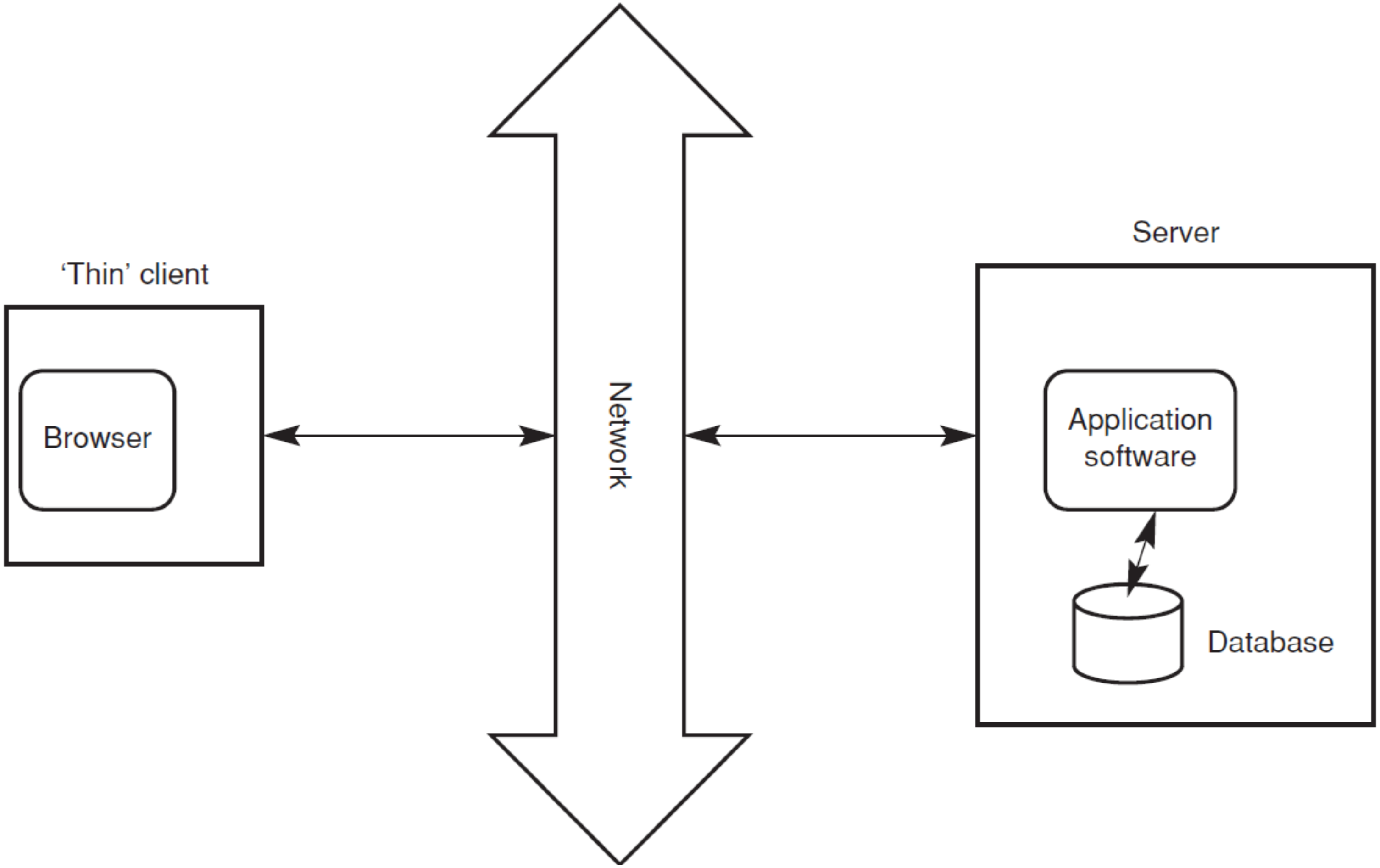


Category	Characteristics	Examples of Styles
Data-flow	Motion of data, with no ‘upstream content control’ by the recipient	Batch sequential Pipe-and-filter
Call-and-return	Order of computation with a single thread of control	Main program/subprograms ‘Classical’ Objects
Interacting processes	Communication among independent, concurrent processes	Communicating processes Distributed Objects
Data-centred repository	Complex central data store	Transactional databases Client–Server Blackboard
Data-sharing	Direct sharing of data among components	Hypertext Lightweight threads

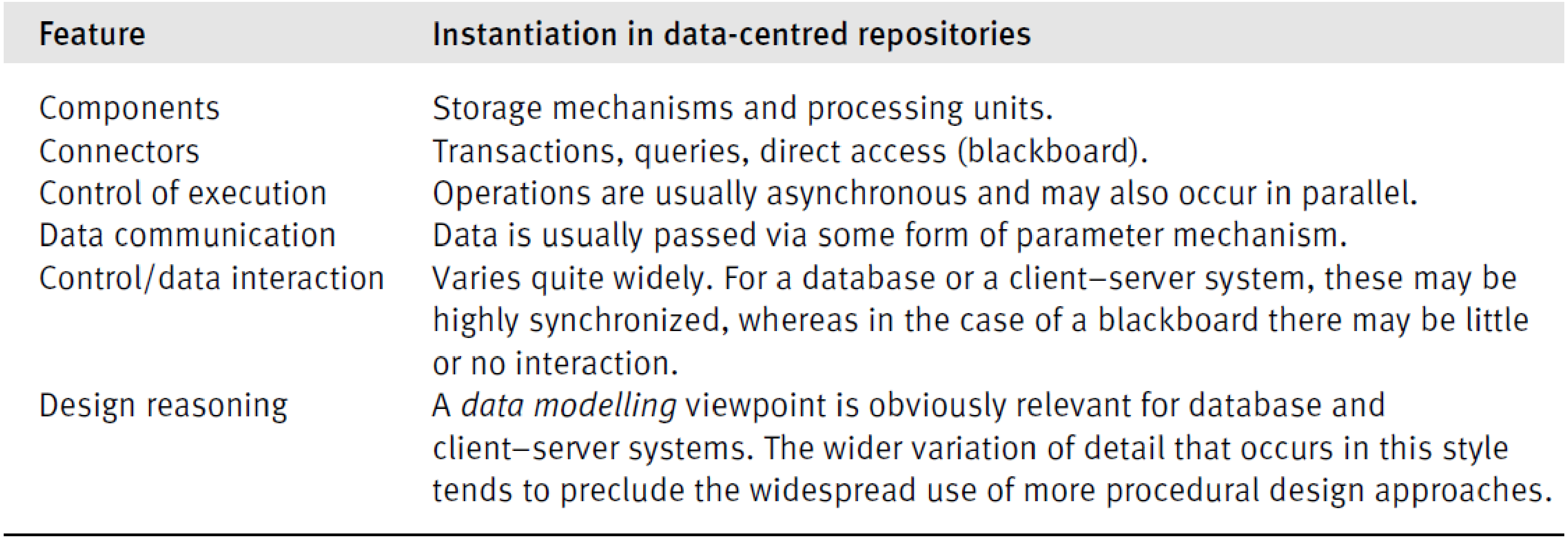
Example: Data-centred repository

- Some central mechanism used for the persistent storage of information which can then be manipulated independently by some arbitrary number of processing units
- Database systems, blackboard expert systems, and client–server forms come into this general category
- In blackboard systems, control and sequencing of updates is loosely controlled

Example: Data-centred repository



Example: Data-centred repository



Feature	Instantiation in data-centred repositories
Components	Storage mechanisms and processing units.
Connectors	Transactions, queries, direct access (blackboard).
Control of execution	Operations are usually asynchronous and may also occur in parallel.
Data communication	Data is usually passed via some form of parameter mechanism.
Control/data interaction	Varies quite widely. For a database or a client–server system, these may be highly synchronized, whereas in the case of a blackboard there may be little or no interaction.
Design reasoning	A *data modelling* viewpoint is obviously relevant for database and client–server systems. The wider variation of detail that occurs in this style tends to preclude the widespread use of more procedural design approaches.

The role of the architectural concept in knowledge transfer

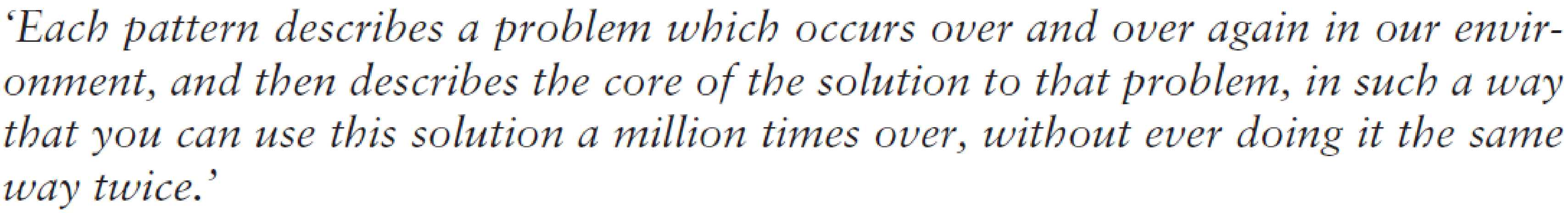
- Providing a framework and vocabulary for top-level design ideas.
- Determining the choice of design strategy.
- Assisting with later changes.

Design methods

- Architecture: codifying knowledge about the *form* of a design solution
- Method: codifying knowledge about *how* to generate design solutions

- Providing a *procedural description* of how to set about the task of producing a design solution for a given problem

Design patterns



‘Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.’

Christopher Alexander et al., 1977, A pattern language

Design patterns

- A pattern is a proven solution to a problem type in a context.

- Design patterns represent a solutions to problems that arise when developing software within a particular context.

Patterns = (problem, solution) pairs in a context

Example: MVC Design Pattern

- **Name** (essence of the pattern)
  - Model View Controller MVC

- **Context** (where does this problem occur)
  - MVC is an **architectural pattern** that is used when developing interactive application such as a shopping cart on the Internet.

- **Problem** (definition of the re-occurring difficulty)
  - User interfaces change often, especially on the internet where look-and-feel is a competitive issue. Also, the same information is presented in different ways.
  - The core business logic and data is stable.

Example: MVC Design Pattern

- **Solution** (how do you solve the problem)
  - Use the software engineering principle of "**separation of concerns**" to divide the application into three areas:
    - **Model** encapsulates the core data and functionality
    - **View** encapsulates the presentation of the data there can be many views of the common data
    - **Controller** accepts input from the user and makes request from the model for the data to produce a new view.

Summary

- We examined some of the principal ways in which software design knowledge and expertise can be codified and transferred:
- the reasons for transferring design knowledge,
- the role of the concept of architectural style in providing a framework and a vocabulary for top-level design ideas,
- the use of design methods to codify design practices and strategies;
- the rationale for using design patterns and their ability to describe the core features of reusable design solutions.

Questions?