

Chapter 9

■ Architectural Design

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e
by Roger S. Pressman

Slides copyright © 1996, 2001, 2005, 2009 by Roger S. Pressman

For non-profit educational use only

May be reproduced ONLY for student use at the university level when used in conjunction with *Software Engineering: A Practitioner's Approach, 7/e*. Any other reproduction or use is prohibited without the express written permission of the author.

All copyright information MUST appear if these slides are posted on a website for student use.

Why Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

- (1) **analyze the effectiveness of the design** in meeting its stated requirements,
- (2) **consider architectural alternatives** at a stage when making design changes is still relatively easy, and
- (3) **reduce the risks** associated with the construction of the software.

Why is Architecture Important?

- **Representations of software architecture are an enabler** for communication between all parties (stakeholders) interested in the development of a computer-based system.
- **The architecture highlights early design decisions** that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- **Architecture “constitutes a relatively small, intellectually graspable mode** of how the system is structured and how its components work together” [BAS03].

Architectural Descriptions

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive System*, [IEE00]
 - to establish a conceptual framework and vocabulary for use during the design of software architecture,
 - to provide detailed guidelines for representing an architectural description, and
 - to encourage sound architectural design practices.
- The IEEE Standard defines an *architectural description* (AD) as a “a collection of products to document an architecture.”
 - The description itself is represented using multiple views, where each *view* is “a representation of a whole system from the perspective of a related set of [stakeholder] concerns.”

Architectural Genres

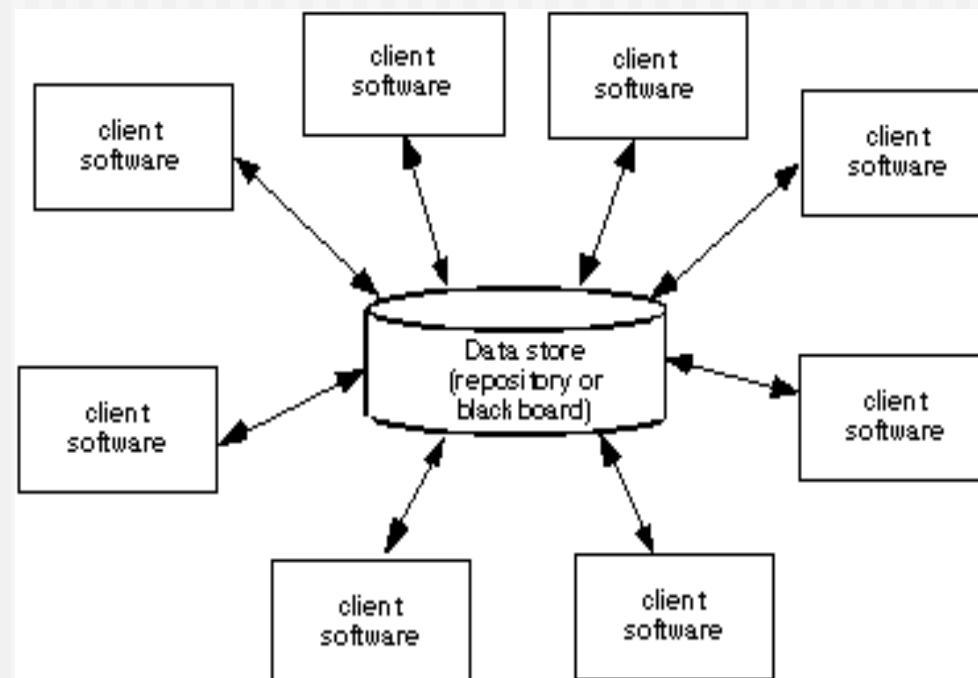
- *Genre* implies a specific category within the overall software domain.
- Within each category, you encounter a number of subcategories.
 - For example, within the genre of *buildings*, you would encounter the following general *styles*: houses, condos, apartment buildings, office buildings, industrial building, warehouses, and so on.
 - Within each general style, more specific styles might apply. Each style would have a structure that can be described using a set of predictable patterns.

Architectural Styles

Each 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, coordination and cooperation” among components, (3) **constraints** that define how components can be integrated to form the system, and (4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

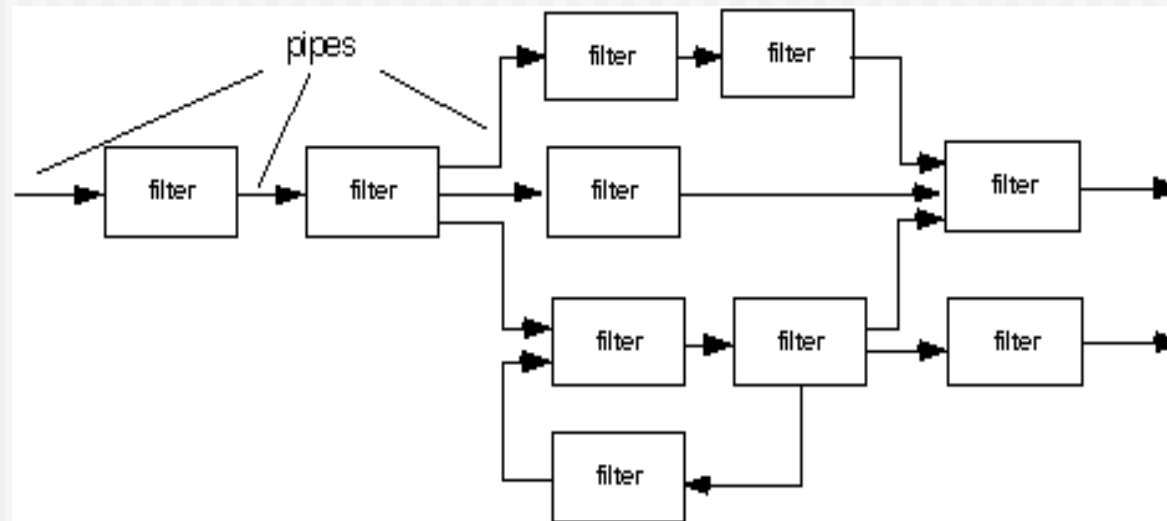
- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

Data-Centered Architecture



These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.

Data Flow Architecture



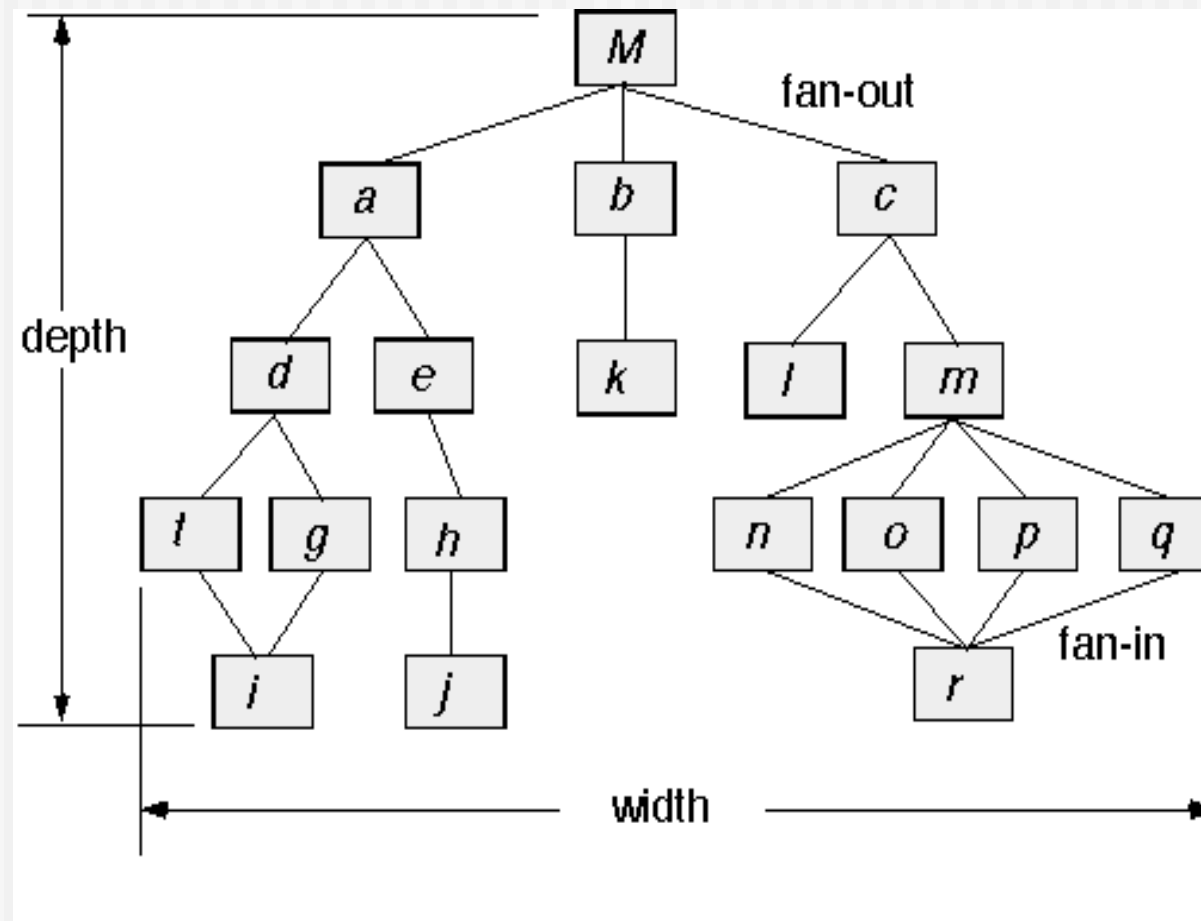
(a) pipes and filters



(b) batch sequential

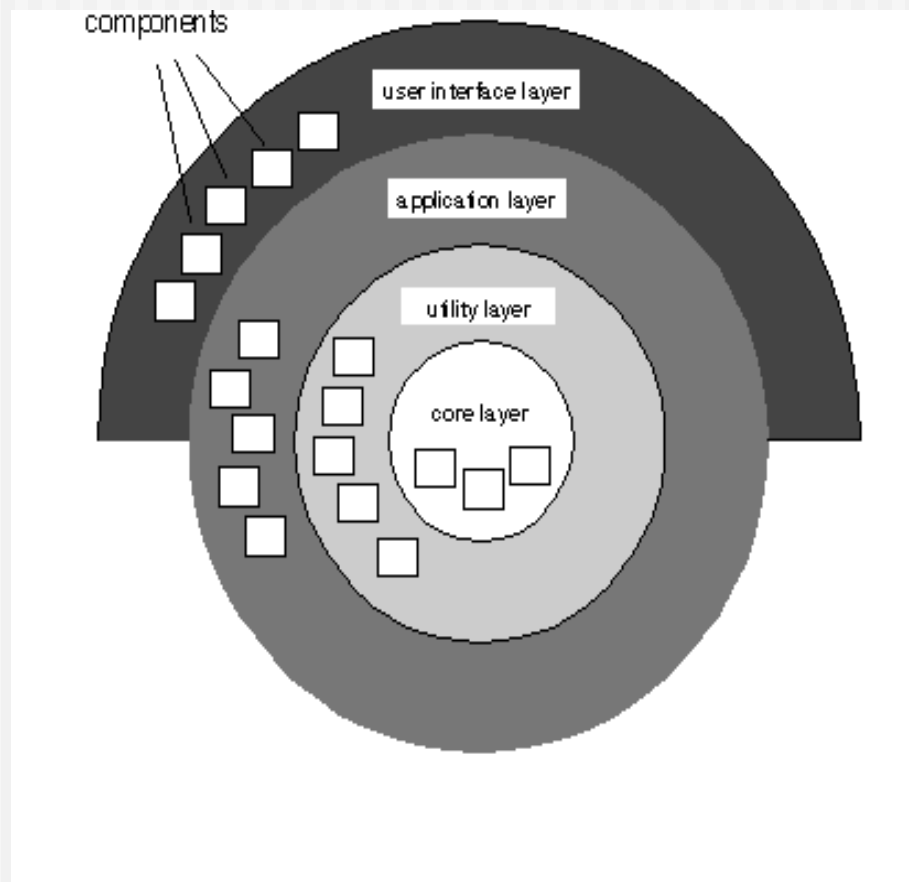
These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.

Call and Return Architecture



These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.

Layered Architecture



These slides are designed to accompany *Software Engineering: A Practitioner's Approach*, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.

Architectural Patterns

- **Concurrency**—applications must handle multiple tasks in a manner that simulates parallelism
 - *operating system process management* pattern
 - *task scheduler* pattern
- **Persistence**—Data persists if it survives past the execution of the process that created it. Two patterns are common:
 - a *database management system* pattern that applies the storage and retrieval capability of a DBMS to the application architecture
 - an *application level persistence* pattern that builds persistence features into the application architecture
- **Distribution**— the manner in which systems or components within systems communicate with one another in a distributed environment
 - A *broker* acts as a ‘middle-man’ between the client component and a server component.