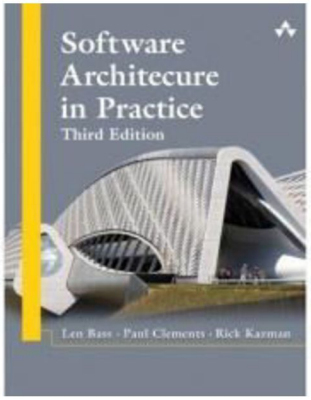
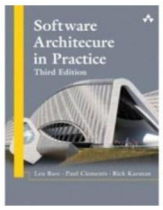


3
Software Architectural Patterns



Modules vs. Components

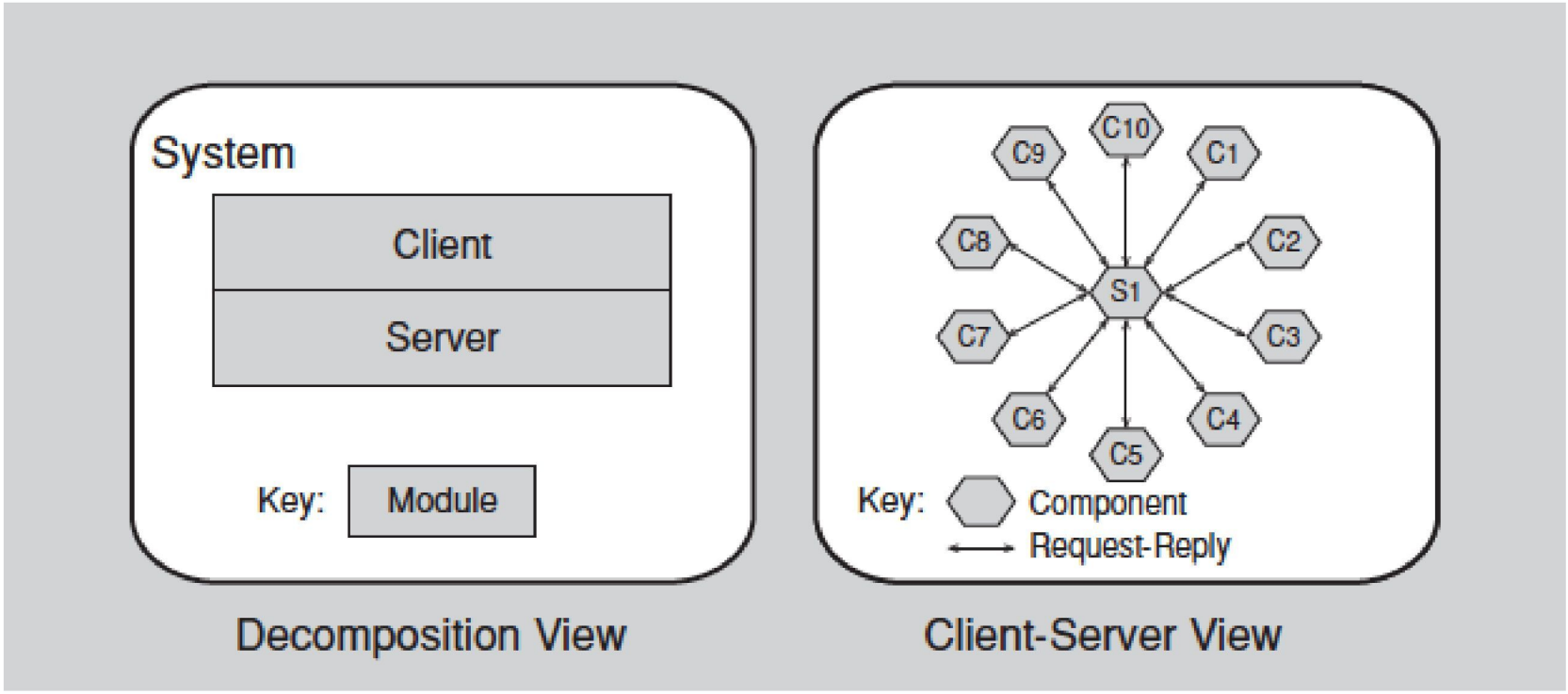
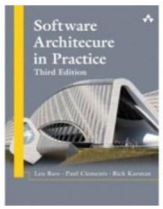
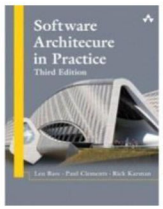


FIGURE 1.2 Two views of a client-server system



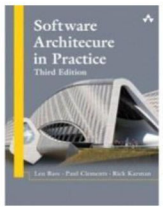
Architectural Patterns

- Architectural elements can be composed in ways that solve particular problems.
  - The compositions have been found useful over time, and over many different domains
  - They have been documented and disseminated.
  - These compositions of architectural elements, called architectural patterns.
  - Patterns provide packaged strategies for solving some of the problems facing a system.
- An architectural pattern delineates the element types and their forms of interaction used in solving the problem.



Architectural Patterns

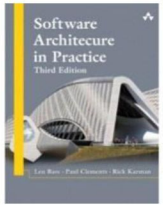
- A common module type pattern is the Layered pattern.
  - When the uses relation among software elements is strictly unidirectional, a system of layers emerges.
  - A layer is a coherent set of related functionality.
  - Many variations of this pattern, lessening the structural restriction, occur in practice.



Architectural Patterns

Common component-and-connector type patterns:

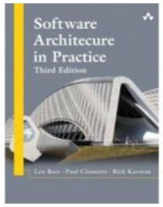
- Shared-data (or repository) pattern.
  - This pattern comprises components and connectors that create, store, and access persistent data.
  - The repository usually takes the form of a (commercial) database.
  - The connectors are protocols for managing the data, such as SQL.
- Client-server pattern.
  - The components are the clients and the servers.
  - The connectors are protocols and messages they share among each other to carry out the system's work.



Architectural Patterns

Common allocation patterns:

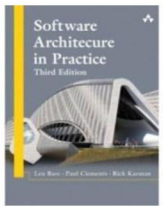
- Multi-tier pattern
  - Describes how to distribute and allocate the components of a system in distinct subsets of hardware and software, connected by some communication medium.
  - This pattern specializes the generic deployment (software-to-hardware allocation) structure.
- Competence center pattern and platform pattern
  - These patterns specialize a software system's work assignment structure.
  - In competence center, work is allocated to sites depending on the technical or domain expertise located at a site.
  - In platform, one site is tasked with developing reusable core assets of a software product line, and other sites develop applications that use the core assets.



What Makes a "Good" Architecture?

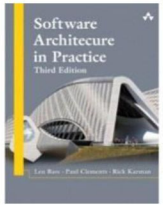
- There is no such thing as an inherently good or bad architecture.
- Architectures are either more or less fit for some purpose
- Architectures can be evaluated but only in the context of specific stated goals.

- There are, however, good rules of thumb.



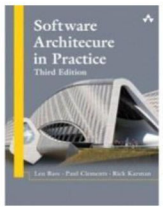
Process "Rules of Thumb"

- The architecture should be the product of a single architect or a small group of architects with an identified technical leader.
  - This approach gives the architecture its conceptual integrity and technical consistency.
  - This recommendation holds for Agile and open source projects as well as "traditional" ones.
  - There should be a strong connection between the architect(s) and the development team.



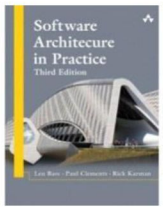
Process “Rules of Thumb”

- The architect (or architecture team) should base the architecture on a prioritized list of well-specified quality attribute requirements.
- The architecture should be documented using views. The views should address the concerns of the most important stakeholders in support of the project timeline.



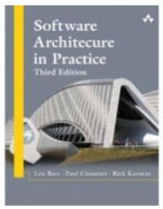
Process "Rules of Thumb"

- The architecture should be evaluated for its ability to deliver the system's important quality attributes.
  - This should occur early in the life cycle and repeated as appropriate.
- The architecture should lend itself to incremental implementation.
  - Create a "skeletal" system in which the communication paths are exercised but which at first has minimal functionality.



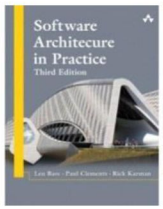
Structural "Rules of Thumb"

- The architecture should feature well-defined modules whose functional responsibilities are assigned on the principles of information hiding and separation of concerns.
  - The information-hiding modules should encapsulate things likely to change
  - Each module should have a well-defined interface that encapsulates or "hides" the changeable aspects from other software



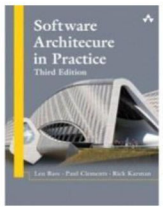
Structural "Rules of Thumb"

- Unless your requirements are unprecedented your quality attributes should be achieved using well-known architectural patterns and tactics specific to each attribute.
- The architecture should never depend on a particular version of a commercial product or tool. If it must, it should be structured so that changing to a different version is straightforward and inexpensive.



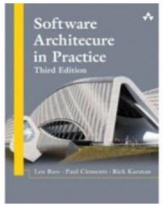
Structural "Rules of Thumb"

- Modules that produce data should be separate from modules that consume data.
  - This tends to increase modifiability
  - Changes are frequently confined to either the production or the consumption side of data.



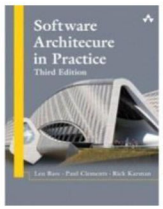
Structural “Rules of Thumb”

- Don’t expect a one-to-one correspondence between modules and components.
- Every process should be written so that its assignment to a specific processor can be easily changed, perhaps even at runtime.



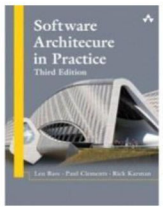
Structural "Rules of Thumb"

- The architecture should feature a small number of ways for components to interact.
  - The system should do the same things in the same way throughout.
  - This will aid in
  - understandability,
  - reduce development time,
  - increase reliability, and
  - enhance modifiability.



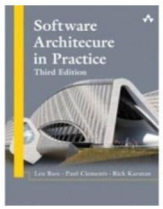
Structural “Rules of Thumb”

- The architecture should contain a specific (and small) set of resource contention areas, the resolution of which is clearly specified and maintained.



Summary

- The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.
- A structure is a set of elements and the relations among them.
- A view is a representation of a coherent set of architectural elements. A view is a representation of one or more structures.



Summary

- There are three categories of structures:
  - Module structures show how a system is to be structured as a set of code or data units that have to be constructed or procured.
  - Component-and-connector structures show how the system is to be structured as a set of elements that have runtime behavior (components) and interactions (connectors).
  - Allocation structures show how the system will relate to nonsoftware structures in its environment (such as CPUs, file systems, networks, development teams, etc.).
- Structures represent the primary engineering leverage points of an architecture.
- Every system has a software architecture, but this architecture may be documented and disseminated, or it may not be.
- There is no such thing as an inherently good or bad architecture. Architectures are either more or less fit for some purpose.